NPS-55HD72101A

NAVAL POSTGRADUATE SCHOOL
Monterey, California

NATURAL LANGUAGE INPUTS TO A
SIMULATION PROGRAMMING SYSTEM

by

George E. Heidorn

October 1972

NAVAL POSTGRADUATE SCHOOL
Monterey, California

ABSTRACT:

This report describes research which has been done toward developing a system for performing simulation analyses through natural language interaction with a computer. A general system for natural language processing, consisting of an IBM 360 FORTRAN program and a "rule language", has been developed. The user of this system must write sets of "decoding" and "encoding" rules, along with some declarations, to specify how processing is to be done for his application. Decoding rules specify how text is to be processed to produce an entity-attribute-value data structure, and encoding rules specify how text is to be produced from such a data structure. The FORTRAN program does the processing according to the sets of rules it is given. Several sets of decoding and encoding rules have been written to implement a specific system which is capable of carrying on a dialogue in English about a simple queuing problem and then producing a program in the GPSS language to do the simulation.

This task was partially supported by the Information Systems Program of the Office of Naval Research as Project NR 049-314, under Project Order PO 1-0177.

The facilities of the W. R. Church Computer Center were utilized for this research.

TABLE OF CONTENTS

TABLE OF CONTENTS

Page

TABLE OF CONTENTS

TABLE OF CONTENTS

Page

TABLE OF CONTENTS

Page

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

This report describes research which is being done to develop a system for performing simulation analyses through natural language interaction with a computer. The eventual goal is to enable an analyst seated at a terminal to "talk" with the computer about his simulation problem in his own natural language (e.g. English), and have the computer "understand" the problem and do the simulation, reporting the results in the same natural language. In such a system the computer would function as an intelligent assistant with great computing powers - actually a combination programmer and computer. This could be considered to be a particular version of the "computer assistant" proposed by Yershov [46].

The objectives of this research are actually twofold, with the system described above intended to be just a particular application of a more general system which could be used for a wide variety of natural language processing tasks. This research can be viewed then either as improving simulation methodology by applying techniques of computational linguistics, or as improving techniques of computational linguistics by applying them to simulation methodology. The approach taken has been to develop techniques just as they are needed for the simulation analysis system described above but to do this development in such a way that many of these techniques can be more generally applicable.

Part of what has been developed is a general system for natural language processing, consisting of an IBM 360 FORTRAN program and a "rule language". The user of this system must write sets of "decoding" and "encoding" rules, along with some declarations, to specify how processing is to be done for his application. Decoding rules specify how text is to be processed to produce an entity-attribute-value data structure, and encoding rules specify how text is to be produced from such a data structure. The FORTRAN program does the processing according to the sets of rules it is given. This general system is referred to as NLP - Natural Language Processor

Also, by writing several sets of rules a specific system has been implemented within NLP toward the goal stated in the opening paragraph. This system is capable of carrying on a dialogue in English about a simple queuing problem and then producing a program in the GPSS language [7] to do the simulation. This is essentially the programmer part of the total simulation analysis system described initially. This specific system, i.e. NLP with these particular sets of rules for processing queuing problems, is referred to as NLPQ.

This chapter begins with a sample problem to demonstrate some of the capabilities of NLPQ and to illustrate the overall manner of processing. Then there is a section which gives an overview of NLP, followed by a brief section about the FORTRAN program. The details of both NLP and NLPQ are presented in Chapters II through IV.

A. A SAMPLE PROBLEM

The sample problem presented in this section is split into two parts. The first part is a complete but simple example which is then used as the basis for discussions about the overall manner of processing. The second part is an expansion of the initial problem to demonstrate the system's capabilities in a highly interactive mode, and shows the system and the user both asking and answering questions.

1. The First Part of the Sample Problem

Figures 1.1, 1.2, and 1.3 show the first part of the terminal session for the sample problem. The statement numbers which appear in these figures and in later ones were added for ease of reference in the discussion. All of the lower case typing was done by the user, and all of the upper case typing was done by the computer.

The first figure shows the problem description being given to the system. As can be seen there, the user entered six statements and then asked if the problem description was complete. The system realized that information about the interarrival times was missing, so it asked a question about that, which the user responded to with a complete sentence. Then the system asked how long the model was to be run, but the user chose to give information about the basic time unit instead, which would have been the subject of the next question. After processing the reply, the system again asked the question about the run time. This time the user responded with the phrase "eight hours". Then the system announced that it had a complete

1. vehicles arrive at a station.

2. the station has just one pump.

3. a vehicle will leave the station immediately after arriving if the length of the line at the pump is not less than two.

4. otherwise, it is serviced there; then it leaves.

5. service times are exponential, with a mean of 5 minutes for cars and 9 minutes for trucks.

6. three quarters of the vehicles are cars and one fourth of them are trucks.

7. is the statement of the problem complete?

8. HOW OFTEN DO THE VEHICLES ARRIVE AT THE STATION?

9. arrivals are normally distributed with a mean of 8 minutes and a standard deviation of 2 minutes.

10. HOW LONG SHALL THE SIMULATION BE RUN?

11. the basic time unit to be used in the model is 30 seconds.

12. HOW LONG SHALL THE SIMULATION BE RUN?

13. eight hours.

14. THE PROBLEM STATEMENT IS COMPLETE.

Figure 1.1. The English Statement of the Problem Initially Being Given to the System

15. describe the problem in english

16. THE VEHICLES ARRIVE AT THE STATION. THE TIME BETWEEN ARRIVALS OF THE VEHICLES AT THE STATION IS NORMALLY DISTRIBUTED, WITH A MEAN OF 8 MINUTES AND A STANDARD DEVIATION OF 2 MINUTES. 75 PERCENT OF THE VEHICLES ARE CARS, AND THE REST ARE TRUCKS. AFTER ARRIVING AT THE STATION, IF THE LENGTH OF THE LINE AT THE PUMP IN THE STATION IS LESS THAN 2, THE VEHICLES WILL BE SERVICED AT THE PUMP IN THE STATION. OTHERWISE, THE VEHICLES WILL LEAVE THE STATION. THE TIME FOR THE VEHICLES TO BE SERVICED AT THE PUMP IN THE STATION IS EXPONENTIALLY DISTRIBUTED, WITH A MEAN OF 5 MINUTES FOR THE CARS, AND 9 MINUTES FOR THE TRUCKS. AFTER BEING SERVICED AT THE PUMP IN THE STATION, THE VEHICLES LEAVE THE STATION.

THE SIMULATION IS TO BE RUN FOR 8 HOURS, USING A BASIC TIME UNIT OF 30 SECONDS.

Figure 1.2. The Initial English Problem Description Produced by the System

17. write a gpss program for this problem.

18.
```
        SIMULATE
        RMULT      277,423,715,121,655,531,999,813
 STAT1  EQU        1,F,Q
 PUMP2  EQU        2,F,Q
 CAR2   EQU        2,T
 2      TABLE      M1,1,1,2
 TRUC3  EQU        3,T
 3      TABLE      M1,1,1,2
 1      FUNCTION   RN1,C24
0.0,0.0/.100,.104/.200,.222/.300,.355/
.400,.509/.500,.690/.600,.915/.700,1.200/
.750,1.390/.800,1.600/.840,1.830/.880,2.120/
.900,2.300/.920,2.520/.940,2.810/.950,2.990/
.960,3.200/.970,3.500/.980,3.900/.990,4.600/
.995,5.300/.998,6.200/.999,7.000/1.000,8.000/
 2      FUNCTION   RN2,C29
0.0,-3.000/.012,-2.250/.027,-1.930/.043,-1.720/
.062,-1.540/.084,-1.380/.104,-1.260/.131,-1.120/
.159,-1.00/.187,-.890/.230,-.740/.267,-.620/
.334,-.430/.432,-.170/.500,0.0/.568,.170/
.666,.430/.732,.620/.770,.740/.813,.890/
.841,1.000/.869,1.120/.896,1.260/.916,1.380/
.938,1.540/.957,1.720/.973,1.930/.988,2.250/
1.000,3.000/
 3      FUNCTION   P1,D2
CAR2,10/TRUC3,18/
 4      FUNCTION   RN3,D2
.750,CAR2/1.000,TRUC3/
 1      FVARIABLE  16+4*FN2
*
*       THE VEHICLES ARRIVE AT THE STATION.
        GENERATE   V1
        ASSIGN     1,FN4
        TEST L     Q$PUMP2,2,ACT2
        TRANSFER   ,ACT3
*
*       THE VEHICLES LEAVE THE STATION.
 ACT2   TABULATE   P1
        TERMINATE
*
*       THE VEHICLES ARE SERVICED AT THE PUMP.
 ACT3   QUEUE      PUMP2
        SEIZE      PUMP2
        DEPART     PUMP2
        ADVANCE    FN3,FN1
        RELEASE    PUMP2
        TRANSFER   ,ACT2
*
*       TIMING LOOP
        GENERATE   960
        TERMINATE  1
        START      1
        END
```

Figure 1.3. The Initial GPSS Program Produced by the System

problem statement, i.e. one for which it could produce a GPSS program.

In the second figure there is a description of the problem "in the computer's own words", which the user can read carefully to see if the system "understood" him. Then the third figure shows the GPSS program produced by the system for performing the simulation.

2. <u>The Overall Approach</u>

If a simulation programmer were given a queuing problem stated in a natural language, he would probably read it one or more times to form a "mental image" of the system being described and to note the points of interest in it. If the description were not clear to him or if essential information were missing, he might ask questions of the writer until he felt that he completely understood the problem and had all the information he needed to do the program. At this point he might state the problem "in his own words" to the writer as a check on his understanding of it. Finally, he would think about the problem in terms of the concepts of the computer language he planned to use, and then he would write the program.

The computer system being described here serves the same role as the simulation programmer described above. Therefore, it was designed to follow essentially the same overall procedure as he does, as can be seen from the example in Figures 1.1 through 1.3. The computer's counterpart of the programmer's

mental image, the Internal Problem Description, is central to the operation of this system and will be discussed first, followed by discussions of the English input, the English output, and the GPSS program.

3. The Internal Problem Description

The Internal Problem Description (IPD) is an entity-attribute-value data structure for holding information about a particular problem in a language-independent form. Entity-attribute-value data structures have been widely used both in artificial intelligence applications and in simulation programming systems such as SIMSCRIPT [24] and GPSS [7]. In the IPD an entity is represented by a "record", which is just a list of attribute-value pairs. Some of the records in an IPD represent physical entities, such as a car or a dock, and others represent abstract entities, such as an action or a function. The attributes which a record has depend, of course, upon the entity being represented. The value of an attribute may simply be a number or a name, or it may be a pointer to another record.

A queuing problem typically deals with physical entities, such as cars or ships, moving through a system to be serviced in some manner at other physical entities, such as a pump or a dock, in the system. Here, the former of these are termed "mobile entities", and the latter are called "stationary entities". (In SIMSCRIPT these are temporary and permanent entities, and in GPSS they are transactions and facilities

and storages.) As the mobile entities move through the system, they engage in "actions" at the stationary entities. Some of these actions are instantaneous, such as arrive and leave, and are called "events"; others, such as service and load, consume time and are referred to here as "activities".

The IPD describes the flow of mobile entities through a system, by specifying the actions which take place there and their interrelationships. Each of these actions is represented by a record which has attributes to furnish such information as the type of action, the entity doing the action (i.e. the agent), the one to whom the action is being done (i.e. the goal), the location where it happens, how long it takes, how often it occurs, and what happens next. For example, the action "The men unload the ship at a dock for eight hours" could be represented by the record

Type	unload
Agent	men
Goal	ship
Location	dock
Duration	8 hours

where the values of at least some of these attributes are actually pointers to other records in the IPD, such as records to represent the men, ships, and docks. If an attribute such as duration were specified as a probability distribution, there would be a record in the IPD to represent that particular distribution, also.

Each record in the IPD is a member of some list, such as the "action list" or the "mobile entity list". These lists furnish a means for "looking at" all records of a particular type during processing. For instance, when English input is being decoded, items in the input can be "looked up" on the appropriate lists to determine what they refer to. When English and GPSS output are being encoded, these lists provide an organizing mechanism, as will be seen in the discussions in subsections 5 and 6.

A detailed explanation of the form of the Internal Problem Description is given in Chapter II. Included there in Figure 2.8 is a graphic portrayal of the IPD built by the system for the sample problem of Figure 1.1.

4. The English Statement of the Problem

At the beginning of a session with the system there is no IPD. It is the English input that furnishes the information which enables the system to build one. For example, when sentence 1 of Figure 1.1 was processed, records were created to represent the vehicles, the station, and the action "vehicles arrive at the station". Each of these three records was put on the appropriate list for its type. Similarly, when sentence 2 was processed, a record was created for the pump, and that record was added to the stationary entity list.

At the time a record is created, some of its attributes are given values. Then later in the processing of the input, possibly in another sentence, information becomes available

which makes it possible to add additional attribute values to the record. For example, sentence 3 included information about the successor attribute of the action "vehicles arrive at the station". Similarly, sentence 5 provided a value for the duration attribute of the action "vehicles are serviced at the pump".

Sentences which occur in natural language descriptions of queuing problems can be considered to fall into two categories: "action sentences" and "attribute sentences". An action sentence has as its main verb an action verb, which is modified by phrases and clauses to specify the values of the attributes of the action. For example,

> After arriving, if the dock is available,
> the ship is unloaded at the dock.

is an action sentence; the action is "unload", its goal is "ship", its location is "dock", its predecessor is "arrive", and its condition is "dock available". It should be noted that the order of most of the phrases and clauses in this sentence could be changed without altering the information content.

An attribute sentence has as its main verb an attribute verb, and is used to specify the value of some attribute of some record in the IPD. For example,

> The time to unload the ship is 8 hours.

says that the value of the "time" (actually duration) attribute of the action record "unload ship" is "8 hours". An

equivalent statement would be

> It takes 8 hours to unload the ship.

The English statement of the problem must describe the flow of mobile entities through the queuing system. This is done by saying something about each action that takes place there, and how it is related to other actions. Each mobile entity must "arrive" at or "enter" the system. Then it may go through one or more other actions, such as "service", "load", "unload", and "wait". Then, typically, it "leaves" the system. The order in which these actions take place must be made explicit by the use of subordinate clauses beginning with such conjunctions as "after", "when", and "before", or by using the adverb "then". If the order of the actions depends on the state of the system being simulated, an "if" clause may be used to specify the condition for performing an action. Then a sentence with an "otherwise" in it would be used to give an alternate action to be performed when the condition is not met.

In the sample problem in Figure 1.1, sentences 1, 3, and 4 are action sentences describing the flow of vehicles through the station. This same information could be given in a wide variety of ways. For example, the following would be acceptable as input:

> Arrivals of the vehicles occur at a station. If the length of the line at the pump in the station is less than 2 when a vehicle arrives, it will be serviced at the pump. Otherwise, it will leave immediately. After being serviced, a vehicle leaves the station.

In addition to describing the flow of mobile entities through the queuing system, the English statement of the problem must also furnish other information needed to simulate the system, such as the various times involved. It is necessary to specify the time between arrivals, the time required to perform each activity, the length of the simulation run, and the basic time unit to be used in the GPSS program. Also, the quantity of each stationary entity should be specified. (A quantity of one is assumed otherwise.) Other information, such as that of sentence 6 in the example, may also be given.

In the sample problem, sentences 2, 5, 6, 9 and 11 are attribute sentences furnishing this additional information. Just as with the action sentences, this information could be given in a wide variety of ways. For example, sentence 2, which specifies the values of both the location and quantity attributes of the pump, could be stated in at least the following three ways:

> There is one pump in the station.
> The quantity of pumps in the station is one.
> There is one pump, and the pump is in the station.

Each attribute sentence is essentially of the form, "<u>attribute</u> of <u>entity</u> equals <u>value</u>". The name of the <u>attribute</u> may be given explicitly, as is "quantity" in the second sentence above, or it may be implied by the verb or the type of value or some other characteristic of the sentence. For example, the verb "hold" implies the capacity attribute in the following sentence:

> The station can hold three cars.

An equivalent statement would be:

> The capacity of the station is three cars.

In each of the following, the attribute is implied by the type of value:

> The pump is in the station.
> The pump is green.

These are equivalent to:

> The pump is located in the station.
> The color of the pump is green.

In sentence 9 of the sample problem, the attribute "inter-event time" is implied.

The <u>entity</u> referred to may be a physical entity, such as pump and station in the above examples, or it may be an abstract entity, such as an action or a probability distribution. When it is an action, the infinitive or present participle form of the action verb, along with appropriate modifiers, is usually used to identify the action. For example, the following four sentences are all equivalent:

> The time <u>to service a vehicle at the pump</u> is 5 minutes.
> The time for <u>servicing a vehicle at the pump</u> is 5 minutes.
> The time for <u>a vehicle's servicing at the pump</u> is 5 minutes.
> <u>The servicing of a vehicle at the pump</u> takes 5 minutes.

Any one of the first three underlined phrases could have appeared in the fourth sentence of this example, too.

The <u>value</u> part of an attribute sentence can take many different forms, as can be seen in the sample problem and in the above examples. Especially important in simulation models are

quantitative values, of which there are several forms. The following phrases are examples of quantitative values:

ten tons
from 10 to 20 minutes
9 minutes for trucks and 5 minutes for cars
exponentially distributed with a mean of two hours

In the last phrase above there is actually a second level of attribute and value, i.e. the mean (attribute) of the exponential distribution (entity) is two hours (value). This can also be seen in sentences 5 and 9 of the sample problem.

Whenever some action is referenced in either an action sentence or an attribute sentence, only enough modifiers to distinguish that action from others have to be used. For example, sentence 9 of the sample problem could have begun, "Arrivals of vehicles" or "Arrivals of vehicles at the station". However, it was sufficient to simply say, "Arrivals", because only one "arrive" action had been previously mentioned in the problem description.

Some use of pronominal reference is allowed, also. For instance, "it" is considered to refer to the most recent non-person mobile entity or stationary entity mentioned, whichever would make a meaningful sentence in the queuing problem context. Similarly, "there" is considered to be a substitute for the most recent location phrase. Both of these can be seen in sentence 4 of the sample problem, where "it" refers to the "vehicle" and "there" means "at the pump". The following sentence would actually have exactly the same meaning as sentence 4:

Otherwise, it is serviced at it; then it leaves.

In this case the middle "it" would be taken as "the pump", because a location phrase requires a <u>stationary</u> entity in the queuing problem context.

Most sentences of the input text are completely parsed (at least implicitly) by the decoding process, i.e. every word and phrase must be accounted for. However, the system is also capable of extracting meaning from some sentences just by the appearance of certain "keywords". For instance, if the words "time" and "unit" and some time phrase (e.g. "30 seconds") appear in a sentence, the time phrase is considered to be the basic time unit to be used in the GPSS program. Similarly, the appearance of "GPSS" or "program" results in the GPSS program being produced. In the sample problem, sentences 7, 11, 15 and 17 are keyword sentences.

In the English input the user may either state the complete problem immediately, or he may state just some part of it and then let the system ask questions to obtain the rest of the information, as was done in Figure 1.1. Each time the system asks a question, it is trying to obtain the value of some one essential attribute. A question may be answered by a complete sentence (e.g. statement 9) or simply by an appropriate phrase (e.g. statement 13) to furnish a value for the attribute, or the question may be ignored and a sentence with some other information given (e.g. statement 11). The user may ask questions of the system, also. Many more examples

of this type of interaction will be seen in the continuation of the sample problem discussed in subsection 7.

A detailed explanation of the decoding of English text in this system is given in Chapter IV.

5. The English Problem Description Produced by the System

The first paragraph of the English problem description is produced by going down the action list and saying something about the attributes of each action. The very first action is simply stated with an action sentence containing information about the type of action, its agent and/or goal, and its location. If the inter-event time or duration attribute has a simple constant value, it will be included also, as a prepositional phrase (e.g. "every 8 minutes" or "for 15 minutes"). If the duration attribute has a "conditional" value, it will be included as a subordinate clause beginning with "until". Otherwise, a separate statement in the form of an attribute sentence will be made about the inter-event time or duration, as can be seen in Figure 1.2. If the action has an "assignment distribution", a statement will then be made about it, as also can be seen in the figure ("75 percent of...."). Finally, a statement of the form "After ..., ..." is produced from the successor attribute, with the exact form of this statement depending upon the type of value which the successor attribute has. It can be seen in the figure that this may actually result in two sentences, with the first one having an "if" clause and the second one beginning with "otherwise".

When describing an action which has already been mentioned in a successor statement, it is not necessary to produce a simple action sentence about it. If the action has a non-simple duration and/or a successor attribute, the appropriate statements about these can immediately be made. This is the case for the "service" action in the example. No output was produced from the "leave" action, because it had already been mentioned in a successor statement and it had no additional attributes to be described.

If a stationary entity has a quantity or capacity attribute with a value greater than 1, a statement will be made about it shortly after the entity is first mentioned in an action sentence (e.g. "There are 2 pumps in the station." or "The capacity of the station is 8 vehicles."). After describing the actions and the entities, a separate one-sentence paragraph is produced with the values of the run time and the basic time unit, as can be seen in the figure.

The English problem description for the expanded sample problem in subsection 7 furnishes additional examples of the kinds of statements produced by the system. A detailed explanation of the encoding of English problem descriptions is included in Chapter III.

6. The GPSS Program Produced by the System

The manner of producing the GPSS program is similar to that for the English problem description, but it involves going down several lists, not just the action list. The first

bit of GPSS program produced is a standard SIMULATE card and RMULT card, as can be seen in Figure 1.3. Then a pass is made down the stationary entity list to produce an EQU card for each stationary entity, to relate its name and its identification number and to define it as a facility or a storage and a queue. If either the quantity or capacity attribute is greater than 1, an appropriate STORAGE definition card is also produced.

Then a similar pass is made down the mobile entity list to output an EQU card and a TABLE card for each type of mobile entity that will actually appear in the simulation. In the example, nothing is included for "vehicle" because any vehicle that appears is either a car or a truck. The tables defined will be used to record transit times during the simulation.

Next, a standard FUNCTION 1 for the exponential distribution and a standard FUNCTION 2 for the unit normal distribution are produced if they are required by the problem. Then a pass is made down the distribution list to define a FUNCTION for each record that requires one. In the example, FUNCTION 3 and FUNCTION 4 came about this way. This is followed by a similar pass down the successor descriptor list to define a FUNCTION for each record that requires one. This pass produced nothing in the example.

Then the records in the distribution list are looked at once again to define an FVARIABLE for each normal distribution used in the problem. One of these appears in the example. The numbers 16 and 4 appear there for the mean and standard

deviation rather than 8 and 2, as might be expected, because the basic time unit to be used for this problem was specified as 30 seconds rather than 1 minute.

After the definitions have been taken care of, a pass is made down the action list to produce the executable blocks which describe the flow of transactions through the program (which corresponds to the flow of mobile entities through the actual system). For each action a blank comment card (with an asterisk in column 1), followed by a comment card with a simple action sentence on it is immediately put out. This is then followed by the blocks appropriate to this action.

The group of blocks produced from an action actually has two parts, the first of which depends upon the type of action and the second of which depends upon the type of value its successor attribute has. For example, an "arrive" usually produces a GENERATE and an ASSIGN, a "leave" produces a TABULATE and a TERMINATE, and most activites produce a sequence like QUEUE, SEIZE, DEPART, ADVANCE, and RELEASE, or minor variations thereof. A "queue-type" successor results in a TEST, followed by a TRANSFER (if necessary), and a simple successor results in an unconditional TRANSFER, as can be seen in the example. If the "leave" and "service" actions had been in reverse order in the action list, the resulting GPSS program would not have needed the two unconditional TRANSFER's which appear in this program, and they would have been suppressed.

The contents of most of the argument fields of the various blocks depend, of course, upon the attributes of the records in the IPD. For example, argument A of the GENERATE block is V1 here because FVARIABLE 1 corresponds to the normal distribution which is the value of the inter-event time attribute of the "arrive" action. Similarly, argument B of the ASSIGN block (which assigns the transaction type, either 2 or 3, to parameter 1 of the transaction) comes from the "assignment distribution" attribute of the same action. Arguments A, B, and C of the TEST block and argument B of the TRANSFER come directly from the attributes of the successor descriptor record for the "arrive". The location attribute of an action record determines the A argument for such blocks as QUEUE, DEPART, SEIZE, and RELEASE, as can be seen in the example.

It can also be seen that argument A of the ADVANCE block (the mean advance time) references FUNCTION 3, which was defined from the record which specifies the mean of the duration of the "service" action. When a transaction enters that ADVANCE block, the appropriate mean time will be obtained from FUNCTION 3 using the value of parameter 1 which was ASSIGN'ed to it when it "arrived". This will then be modified by a value from FUNCTION 1 to yield a service time from the desired exponential distribution. The B argument of the last TRANSFER gets its value directly from the successor attribute of the "service" action. All actions are referenced by names of the form "ACTi", where i is the value of the action's identification number.

Finally, after the blocks for the actions are put out, a standard "timing loop" is produced to govern the run length of the simulation. The value in the A argument of the GENERATE block comes from the run time specified. In the example this value is 960, because there are 960 30-second periods in 8 hours.

Another GPSS program which is longer than this example, but which has exactly the same overall structure of course, will be seen for the expanded sample problem in the next sub-section. A detailed explanation of the encoding of GPSS programs is also included in Chapter III.

7. <u>The Remainder of the Sample Problem</u>

The remainder of the terminal session for the sample problem is shown in Figures 1.4, 1.5, and 1.6. The first of these, which is three pages long, shows the problem being expanded and modified through interaction with the system. The second one shows the new English problem description produced by the system, and the last one, which is two pages long, shows the new GPSS program produced. As in the earlier figures, all of the lower case typing was done by the user, and all of the upper case typing was done by the system.

The first sentence (19) given in Figure 1.4 primarily had the effect of changing the successor attribute of the "service" action in the original problem from "vehicles leave the station" to "cars leave the station but trucks unload cargo", resulting in an incomplete problem specification. Statement 20 is a keyword sentence which put the system into

19. cars leave the station after being serviced, but trucks unload cargo.

20. question me for necessary info.

21. WHERE DO THE TRUCKS UNLOAD CARGO?

22. at a dock.

23. HOW LONG DO THE TRUCKS UNLOAD CARGO AT THE DOCK?

24. it takes ten minutes for the cargo to be unloaded.

25. AFTER UNLOADING CARGO AT THE DOCK, WHAT DO THE TRUCKS DO?

26. one half of them leave the station, and a third wait in the depot until the pier is available.

27. THE FOLLOWING PERCENTAGES DO NOT TOTAL 100: AFTER UNLOADING CARGO AT THE DOCK FOR 10 MINUTES, 50 PERCENT OF THE TRUCKS LEAVE THE STATION, AND 33 PERCENT WAIT IN THE DEPOT UNTIL THE PIER IS AVAILABLE. AFTER UNLOADING CARGO AT THE DOCK, WHAT DO THE TRUCKS DO?

28. half leave and half wait.

29. AFTER WAITING IN THE DEPOT UNTIL THE PIER IS AVAILABLE, WHAT DO THE TRUCKS DO?

30. they unload at the pier if the pump is free.

31. OTHERWISE, WHAT DO THE TRUCKS DO?

32. what?

33. AFTER WAITING IN THE DEPOT UNTIL THE PIER IS AVAILABLE, IF THE PUMP IS NOT BUSY, THE TRUCK WILL UNLOAD AT THE PIER. OTHERWISE, WHAT DO THE TRUCKS DO?

34. cargo will be loaded for 15 minutes by the pier by a truck.

35. HOW LONG DO THE TRUCKS UNLOAD AT THE PIER?

36. how long do they unload at a dock?

37. THE TRUCKS UNLOAD CARGO AT THE DOCK FOR 10 MINUTES. HOW LONG DO THE TRUCKS UNLOAD AT THE PIER?

38. unloading at the pier is exponentially distributed.

Figure 1.4. Expansion and Modification of the Problem Through Interaction with the System (1/3)

39. THE TIME FOR THE TRUCKS TO UNLOAD AT THE PIER IS EXPONENTIALLY DISTRIBUTED. WHAT IS THE MEAN OF THE EXPONENTIAL DISTRIBUTION?

40. just ask questions, please.

41. WHAT IS THE MEAN OF THE EXPONENTIAL DISTRIBUTION?

42. normal.

43. WHAT IS THE MEAN OF THE NORMAL DISTRIBUTION?

44. 10 pounds.

45. THE TIME FOR THE TRUCKS TO UNLOAD AT THE PIER CAN NOT BE EXPONENTIALLY DISTRIBUTED, WITH A MEAN WHICH IS NORMALLY DISTRIBUTED, WITH A MEAN OF 10 POUNDS, BECAUSE THE UNITS ARE WRONG. WHAT IS THE MEAN OF THE NORMAL DISTRIBUTION?

46. 10 minutes.

47. WHAT IS THE STANDARD DEVIATION OF THE NORMAL DISTRIBUTION?

48. the same as the standard deviation of the time between arrivals of vehicles at the station.

49. AFTER UNLOADING AT THE PIER, WHAT DO THE TRUCKS DO?

50. they leave.

51. AFTER LOADING CARGO BY THE PIER, WHAT DO THE TRUCKS DO?

52. leave.

53. THE PROBLEM STATEMENT IS COMPLETE.

54. are there two docks in the station?

55. NO, THERE IS 1 DOCK IN THE STATION.

56. the station should have two docks.

57. how often does a vehicle arrive?

58. THE TIME BETWEEN ARRIVALS OF THE VEHICLES AT THE STATION IS NORMALLY DISTRIBUTED, WITH A MEAN OF 8 MINUTES AND A STANDARD DEVIATION OF 2 MINUTES.

59. the mean of the time between vehicle arrivals should be 7 minutes.

Figure 1.4. Expansion and Modification of the Problem Through Interaction with the System (2/3)

60. what do trucks do?

61. PLEASE BE MORE SPECIFIC.

62. what do trucks do after they load?

63. AFTER LOADING CARGO BY THE PIER FOR 15 MINUTES, THE TRUCKS LEAVE THE STATION.

64. after leaving, what do trucks do?

65. I DON'T KNOW.

66. where do trucks unload for 10 minutes?

67. THE TRUCKS UNLOAD CARGO AT THE DOCK IN THE STATION FOR 10 MINUTES.

68. it should take from 6 to 12 minutes to unload at a dock.

69. where do trucks unload for 10 minutes?

70. I DON'T KNOW.

71. how long do they unload at a dock?

72. THE TIME FOR THE TRUCKS TO UNLOAD CARGO AT THE DOCK IN THE STATION IS UNIFORMLY DISTRIBUTED, WITH A MEAN OF 9 MINUTES AND A HALF-RANGE OF 3 MINUTES.

73. what is the time for the trucks' unloading at the pier?

74. THE TIME FOR THE TRUCKS TO UNLOAD AT THE PIER IS EXPONENTIALLY DISTRIBUTED, WITH A MEAN WHICH IS NORMALLY DISTRIBUTED, WITH A MEAN OF 10 MINUTES AND A STANDARD DEVIATION OF 2 MINUTES.

75. is the mean of the mean of the time of the unloading of trucks at the pier 10 hours?

76. NO, THE MEAN OF THE NORMAL DISTRIBUTION IS 10 MINUTES.

77. in the depot does a truck wait?

78. YES, THE TRUCKS WAIT IN THE DEPOT UNTIL THE PIER IS AVAILABLE.

79. are vehicles serviced at a pump?

80. YES, THE VEHICLES ARE SERVICED AT THE PUMP IN THE STATION.

81. this time use a basic time unit of 20 seconds.

82. give an english description of the current problem; and write a program for it in gpss.

Figure 1.4. Expansion and Modification of the Problem Through Interaction with the System (3/3)

83. THE VEHICLES ARRIVE AT THE STATION. THE TIME BETWEEN ARRIVALS OF THE VEHICLES AT THE STATION IS NORMALLY DISTRIBUTED, WITH A MEAN OF 7 MINUTES AND A STANDARD DEVIATION OF 2 MINUTES. 75 PERCENT OF THE VEHICLES ARE CARS, AND THE REST ARE TRUCKS. AFTER ARRIVING AT THE STATION, IF THE LENGTH OF THE LINE AT THE PUMP IN THE STATION IS LESS THAN 2, THE VEHICLE WILL BE SERVICED AT THE PUMP IN THE STATION. OTHERWISE, THE VEHICLE WILL LEAVE THE STATION. THE TIME FOR THE VEHICLES TO BE SERVICED AT THE PUMP IN THE STATION IS EXPONENTIALLY DISTRIBUTED, WITH A MEAN OF 5 MINUTES FOR THE CARS, AND 9 MINUTES FOR THE TRUCKS. AFTER BEING SERVICED AT THE PUMP IN THE STATION, THE CARS LEAVE THE STATION, AND THE TRUCKS UNLOAD CARGO AT A DOCK IN THE STATION. THERE ARE 2 DOCKS IN THE STATION. THE TIME FOR THE TRUCKS TO UNLOAD CARGO AT A DOCK IN THE STATION IS UNIFORMLY DISTRIBUTED, WITH A MEAN OF 9 MINUTES AND A HALF-RANGE OF 3 MINUTES. AFTER UNLOADING CARGO AT A DOCK IN THE STATION, 50 PERCENT OF THE TRUCKS LEAVE THE STATION, AND THE REST WAIT IN THE DEPOT UNTIL THE PIER IS AVAILABLE. AFTER WAITING IN THE DEPOT, IF THE PUMP IS BUSY, THE TRUCK WILL LOAD CARGO BY THE PIER FOR 15 MINUTES. OTHERWISE, THE TRUCK WILL UNLOAD AT THE PIER. THE TIME FOR THE TRUCKS TO UNLOAD AT THE PIER IS EXPONENTIALLY DISTRIBUTED, WITH A MEAN WHICH IS NORMALLY DISTRIBUTED, WITH A MEAN OF 10 MINUTES AND A STANDARD DEVIATION OF 2 MINUTES. AFTER UNLOADING AT THE PIER, THE TRUCKS LEAVE THE STATION. AFTER LOADING CARGO BY THE PIER, THE TRUCKS LEAVE THE STATION.

THE SIMULATION IS TO BE RUN FOR 8 HOURS, USING A BASIC TIME UNIT OF 20 SECONDS.

Figure 1.5. The New English Problem Description Produced by the System

```
84.        SIMULATE
           RMULT       277,423,715,121,655,531,999,813
 STAT1     EQU         1,F,Q
 PUMP2     EQU         2,F,Q
 DOCK3     EQU         3,S,Q
 3         STORAGE     2
 DEPO4     EQU         4,F,Q
 PIER5     EQU         5,F,Q
 CAR2      EQU         2,T
 2         TABLE       M1,1,1,2
 TRUC3     EQU         3,T
 3         TABLE       M1,1,1,2
 1         FUNCTION    RN1,C24
0.0,0.0/.100,.104/.200,.222/.300,.355/
.400,.509/.500,.690/.600,.915/.700,1.200/
.750,1.390/.800,1.600/.840,1.830/.880,2.120/
.900,2.300/.920,2.520/.940,2.810/.950,2.990/
.960,3.200/.970,3.500/.980,3.900/.990,4.600/
.995,5.300/.998,6.200/.999,7.000/1.000,8.000/
 2         FUNCTION    RN2,C29
0.0,-3.000/.012,-2.250/.027,-1.930/.043,-1.720
.062,-1.540/.084,-1.380/.104,-1.260/.131,-1.120/
.159,-1.000/.187,-.890/.230,-.740/.267,-.620/
.334,-.430/.432,-.170/.500,0.0/.568,.170/
.666,.430/.732,.620/.770,.740/.813,.890/
.841,1.000/.869,1.120/.896,1.260/.916,1.380/
.938,1.540/.957,1.720/.973,1.930/.988,2.250/
1.000,3.000/
 3         FUNCTION    P1,D2
CAR2,15/TRUC3,27/
 4         FUNCTION    RN3,D2
.750,CAR2/1.000,TRUC3/
 5         FUNCTION    P1,D2
CAR2,ACT4/TRUC3,ACT5/
 6         FUNCTION    RN4,D2
.500,ACT6/1.000,ACT7/
 1         FVARIABLE   21+6*FN2
 2         FVARIABLE   30+6*FN2
*
*          THE VEHICLES ARRIVE AT THE STATION.
           GENERATE    V1
           ASSIGN      1,FN4
           TEST L      Q$PUMP2,2,ACT2
           TRANSFER    ,ACT3
*
*          THE VEHICLES LEAVE THE STATION.
 ACT2      TABULATE    P1
           TERMINATE
*
```

Figure 1.6. The New GPSS Program Produced by the System (1/2)

```
*       THE VEHICLES ARE SERVICED AT THE PUMP.
 ACT3   QUEUE       PUMP2
        SEIZE       PUMP2
        DEPART      PUMP2
        ADVANCE     FN3,FN1
        RELEASE     PUMP2
        TRANSFER    ,FN5
*
*       THE CARS LEAVE THE STATION.
 ACT4   TABULATE    P1
        TERMINATE
*
*       THE TRUCKS UNLOAD CARGO AT A DOCK.
 ACT5   QUEUE       DOCK3
        ENTER       DOCK3
        DEPART      DOCK3
        ADVANCE     27,9
        LEAVE       DOCK3
        TRANSFER    ,FN6
*
*       THE TRUCKS LEAVE THE STATION.
 ACT6   TABULATE    P1
        TERMINATE
*
*       THE TRUCKS WAIT IN THE DEPOT.
 ACT7   QUEUE       DEPO4
        SEIZE       DEPO4
        DEPART      DEPO4
        GATE NU     PIER5
        RELEASE     DEPO4
        GATE NU     PUMP2,ACT9
*
*       THE TRUCKS UNLOAD AT THE PIER.
 ACT8   QUEUE       PIER5
        SEIZE       PIER5
        DEPART      PIER5
        ADVANCE     V2,FN1
        RELEASE     PIER5
        TRANSFER    ,ACT6
*
*       THE TRUCKS LOAD CARGO BY THE PIER.
 ACT9   QUEUE       PIER5
        SEIZE       PIER5
        DEPART      PIER5
        ADVANCE     45
        RELEASE     PIER5
        TRANSFER    ,ACT6
*
*       TIMING LOOP
        GENERATE    1440
        TERMINATE   1
        START       1
        END
```

Figure 1.6. The New GPSS Program Produced by the System (2/2)

a question-asking mode in which it inspects the IPD for missing or erroneous information and then asks questions about one attribute at a time.

The next four lines (21-24) show the first two questions and answers. The answer to question 23 could have been given simply as "ten minutes" or "for ten minutes", or it could have been given as something like "trucks unload for ten minutes". Actually, the first question (21) could have been answered "trucks unload at a dock for ten minutes" or "at a dock; unloading takes ten minutes", and then the next question (23) would not have even been asked.

After obtaining values for the location and duration attributes of the "unload" action, the system asked for information about its successor attribute in line 25. The answer (26) was not acceptable, so a message was given asking the question again (27). The new answer (28) was okay. Even though the percentages in sentence 26 were erroneous, action records for "trucks leave the station" and "trucks wait in the depot until the pier is available" were created at that time.

The first of these two new action records was complete, because a "leave" requires neither a duration nor a successor, but the second needed a successor. Therefore, question 29 was asked. The response (30) had a condition in it, so that another question (31) had to be asked. The response of "what?" in line 32 put the system into a more verbose question-asking mode which prefaces many questions with a statement to help the user pinpoint the source of the question, as can be seen in line 33.

The answer given in sentence 34 illustrates the role that a restricted context can play in dealing with syntactic ambiguity. Clearly, each of the prepositional phrases there modifies the main verb. The first one, "for 15 minutes", is a duration phrase, i.e. it supplies a value for the duration attribute of the action. The second one is a location phrase because a "pier" is a stationary entity, and the third one is an agent phrase because a "truck" is a mobile entity and the sentence is passive. If the word "truck" were replaced by "dock" in that sentence, then the last prepositional phrase would be taken to modify "pier" instead, furnishing a value for its location attribute.

The next 14 lines (35-48) all have to do with providing a value for the duration attribute of the action "trucks unload at the pier". After the system asked the initial question (35), the user asked a question (36). The system replied and asked its own question again (37), to which the user responded with a complete sentence (38). A response as simple as "exponential" would have supplied the same information.

Then the system set out to determine the mean of the exponential distribution (39). In sentence 40 the user requested a return to the brief question-asking mode, which resulted in question 41. The response of "normal" was given (42) to demonstrate the system's capability for handling nested probability distributions.

Next the system set out to determine the parameters of this normal distribution (43). The answer given in line 44

was not acceptable, so a message was given and the question repeated (45). Before the system will produce a GPSS program, all quantitative values at all levels in the IPD are checked to determine that their units are appropriate for the attributes they serve as the values of.

The answer given in line 46 was acceptable, so then the next question (47) was asked. The answer (48) illustrates how an attribute in the IPD can be used to furnish the value for another attribute.

After questions 49 and 51 were answered in lines 50 and 52, the system announced (53) that "THE PROBLEM STATEMENT IS COMPLETE." However, in order to modify some problem information given previously and to demonstrate the system's capability for answering questions, the dialogue continued. As can be seen there, the system is able to handle both "yes-no" type questions and "wh-" type questions. A detailed discussion of this dialogue will not be given, but certain items will be pointed out.

It may be noted that questions 66 and 69 are identical, but resulted in different answers (67 and 70). That is because sentence 68 which intervened changed the relevant information in the IPD. It should also be noted that the phrase "from 6 to 12 minutes" in line 68 is considered to imply a uniform distribution with a mean of 9 minutes and a half-range of 3 minutes.

Another point to note is that in sentence 75 there is a series of four prepositional phrases which begin with "of".

This sentence could be considered to be highly ambiguous <u>syntactically</u>, but when the function of each "of" is taken into account, there is only one reasonable interpretation. Because "mean" and "time" are attribute names, the first three uses of "of" are as in "attribute <u>of</u> entity". The fourth one is the "agentive-of", specifying "truck" as the agent of "unload". This sentence furnishes an example of what might be called "nested attribute designation".

It should be noted that the only sentences after line 52 which had any effect on the contents of the IPD are 56, 59, 68, and 81. The others just asked questions about the IPD. Finally, in line 82 the system was asked to produce a new English problem description and GPSS program from the current IPD.

A detailed explanation of the way the system asks questions and answers questions is included in Chapter III.

The new English problem description appears in Figure 1.5. It may be noted that the first 12 lines are identical to those of figure 1.2 (with the exception of a "7" in place of an "8" in the third line), and then there are several lines with the new information which was added to the problem. It can be seen that the overall structure of the description is the same as the earlier one and that it is as described in subsection 5. It can also be seen that many of these sentences are identical or similar to some which appeared in the dialogue of Figure 1.4.

The new GPSS program is given in Figure 1.6 (two pages). Some portions of it are identical to those of the program in Figure 1.3, but there are also many additional statements. Some points which were mentioned in subsection 6 but not illustrated in the earlier program are illustrated in this one. For example, FUNCTION 5 and FUNCTION 6 were produced from the pass down the successor descriptor list. Some numbers are different than in the earlier program because the basic time unit was changed from 30 seconds to 20 seconds.

8. <u>Computer Time</u>

There are three different times which can be reported for a job run on the CP/CMS time sharing system on an IBM 360/67 computer. The "virtual CPU time" does not include system overhead and is essentially the time that the job would take if run under a batch system. The "total CPU time" includes system overhead, most of which is for paging, and depends somewhat on the current load on the system. "Elapsed time" is the time that the user spends sitting at the terminal and can be very highly dependent on the current load.

For the first part of the sample problem given here the virtual CPU time was 77 seconds for Figure 1.1, 26 seconds for Figure 1.2, and 45 seconds for Figure 1.3, for a total of 148 seconds. The total CPU time was 156 seconds for Figure 1.1, 41 seconds for Figure 1.2, and 65 seconds for Figure 1.3, for a total of 262 seconds. The elapsed time for this part of the problem may vary from one half hour to two hours, depending on the load on the system.

For the dialogue of Figure 1.4 the virtual CPU time for each question-answer pair was in the range of approximately 5 to 15 seconds, with most of them taking less than 10 seconds. The virtual CPU time for producing the English description of Figure 1.5 was 42 seconds and for the GPSS program of Figure 1.6 was 78 seconds. The total CPU time in each case was approximately double the virtual time. Elapsed time can be estimated roughly by multiplying the total CPU time by the number of users on the time sharing system.

B. OVERVIEW

This section is intended to provide an overview of NLP, the general system for natural language processing developed in this research. It begins with a discussion of entity-attribute-value data structures and the kinds of information which they hold in this system. Then decoding and encoding rules and the role they play are discussed. Finally, a brief introduction to stratificational linguistics is given.

1. Entity-Attribute-Value Data Structures

A basic premise of this research is that entity-attribute-value data structures furnish an adequate and convenient method for holding information of the sort being dealt with in this system. As stated earlier, this type of information structure has been widely used in artificial intelligence applications and in simulation programming systems. The generality of such structures has been noted by Knuth [14, pg. 462] (where he

uses the term "node" for entity or record and "field" for attribute):

> "What are the main implications of the subjects treated in this chapter? Perhaps the most important conclusion we can reach is that the ideas we have encountered are not limited to computer programming alone; they apply more generally to everyday life. A collection of nodes containing fields, some of which point to other nodes, appears to be a very good abstract model for structural relationships of all kinds; it shows how we can build up complicated structures from simple ones, and we have seen that corresponding algorithms for manipulating the structure can be designed in a natural manner."

1(a). Types of Information

In this system a large part of the information is about the words and concepts of the relevant domain of discourse (queuing problems, in the current application). For each word there is a record, with the values of the attributes of the record specifying the characteristics of the word. For example, the record for the word "arrive" could have a part-of-speech attribute with the value "verb". It might also have a TYPE attribute with the value "intransitive" to indicate that it is an intransitive verb. For each concept there is a record with attributes to relate it to other concepts. For example, the record for the concept "arrive" could have a SUPerset attribute whose value is a pointer to the record for the concept "event", specifying that "arrive" is a type of "event". In some cases economies of storage can result from using a single record to hold information both about a word and about a concept associated with the word.

Another type of information is that resulting from a particular dialogue (about a specific queuing problem in the current application). A record here represents some specific entity, either physical or abstract, introduced in the dialogue. In NLPQ this collection of inter-connected records comprises an Internal Problem Description, as discussed earlier in Section A.3.

Still another type of information is that about a particular sentence or part of a sentence appearing in a dialogue. For example, if the word "arriving" appears, that part of the sentence can be described by a record with a SUP attribute pointing to the record for "arrive" and a FORM attribute with the value PRESPART, meaning that "arriving" is the present participle form of "arrive". For another example, the phrase "is arriving" can be described by a record with a SUP of "arrive" and a FORM attribute with values PRP3SG and PROG, meaning that "is arriving" is the present tense, third person, singular, progressive form of "arrive".

Clearly, a different set of attributes could be used to furnish the same information. For example, three separate attributes TENSE, PERSON, and NUMBER, with values "present", "third", and "singular", respectively, along with a FORM attribute of PROG, could be used in the example above.

In this research no attempt has been made to find the "optimal" set of attributes, but rather a general framework has been established within which alternative schemes can be

experimented with. The user of NLP (the general system) is free to devise and use any attributes he desires. In developing NLPQ (the specific system for queuing problems) certain attributes were decided upon for the various records involved, but they are subject to revision as the system's capabilities are expanded.

From this discussion, three types of information may be readily distinguished according to its relative permanence. The first type is about words and concepts and may be considered to comprise a long-term memory. This is information which the system needs in order to carry on a dialogue, and it is given to the system initially by means of "named record definitions", as will be described later. The second type is that in the IPD for a specific problem and may be considered to comprise a short-term memory. This is information which the system obtains through a dialogue, and then can later discard. The third type is that about a particular sentence or part of a sentence and may be considered to be held in a temporary or scratchpad memory. This information is needed only while that sentence is being processed, and then it can be erased.

An important feature of NLP is that information of all three types is maintained in exactly the same way, i.e. as records with attributes. This makes it possible to manipulate all of this information in the same manner.

1(b). Indicators

It is notationally convenient in some cases to refer to certain attribute values without explicitly naming the attribute. In the work being reported on here, these attribute values are called "indicators", because typically they indicate that some condition exists. For example, a record describing the word "arriving" could be said to have a PRESPART indicator (or it could be said that in this record the PRESPART indicator is "on", as opposed to being "off".)

Indicators are especially useful for attributes that can take on only two opposite values. For example, instead of having a STATE attribute with possible values "alive" and "dead", the presence of an ALIVE indicator (i.e. ALIVE is "on") could be used to mean "alive" and the absence of that indicator (i.e. ALIVE is "off") could mean "dead" (i.e. "not alive"). It may be recognized that there is some correspondence between the notion of "indicator" as the term is used here and the notion of "feature" as that term is used in much of the linguistics literature.

1(c). Equivalent Information

In an entity-attribute-value data structure the order of the elements is not significant. Abstractly, a record is an unordered collection of attribute—value pairs. For example, the following two records are equivalent in that they contain the same information:

SUP	hit	SUP	hit
AGENT	Mary	GOAL	John
GOAL	John	AGENT	Mary

Order is not significant because each piece of information is explicitly labeled as to its role. This is probably the most important feature of this type of data structure.

In natural language text, however, the order of the elements can be extremely significant. For example, although the following two sentences consist of the same words, they certainly do not contain the same information:

Mary hit John.
John hit Mary.

In the first sentence "Mary" occupies the subject position, "John" occupies the direct object position, and the voice of the verb is active, so Mary is the AGENT of the action and John is the GOAL. In the second sentence their positions in the sentence, and hence their roles in the action, are simply reversed.

In record form the information in the two sentences above would be:

SUP	hit	SUP	hit
AGENT	Mary	AGENT	John
GOAL	John	GOAL	Mary

The position of modifying elements often can be changed without altering the information content of a sentence, however. For example, the following four sentences contain the same information:

Today Mary hit John.
Mary today hit John.
Mary hit today John.
Mary hit John today.

(Although the third one sounds awkward, it would probably be understood by most English-speaking people.) The reason that the position of "today" in the above sentences is not critical is that its only possible function is to specify the value of the TIME attribute of the action being described.

The information in each of the four sentences above could be given by the record:

```
SUP      hit
AGENT    Mary
GOAL     John
TIME     today
```

As another example of different word order carrying equivalent information, the above record could just as well describe the sentence:

Today John was hit by Mary.

where "today" could appear anywhere in that sentence, except between "by" and "Mary".

It should be noted that what is being discussed here is "equivalent information" and not "meaning". However, intuitively it would seem that the attribute-value form comes closer to displaying the meaning of a sentence than the character string does.

2. Decoding and Encoding

Basically what the system developed in this research does is to convert information from one form to another. The two different forms of information that it works with are records and character strings. The process of converting from a character string into an equivalent record structure is called "decoding", and the inverse process is called "encoding". During both decoding and encoding, there are typically many intermediate steps in which records are converted to other records, but the system does not provide for converting character strings directly to other character strings. The underlying philosophy is that information is much more manageable in record form than in character string form.

The processing to be done by this system is specified by sets of rules written in a rule language developed for it. Decoding rules are used to specify how character strings are to be converted to records, and encoding rules are used to specify how records are to be converted to character strings. Either kind of rule can be used to specify how records are to be converted to other records.

A rule consists of a left part and a right part, separated by an arrow. In general, the left part specifies a situation which must exist in order to apply the rule, and the right part tells what to do when the rule can be applied. Control over the application of these rules lies with the decoding and encoding algorithms.*

---

* The term "algorithm" as it is used in this report is synonymous with "procedure". It is often used this way in the computational linguistics literature, although many mathematicians would prefer a more restricted use of the term (e.g. [14, pp 1-9]).

2(a). <u>Decoding Rules</u>

The following are examples of decoding rules:

```
# A R R I V --> VERBSTEM('ARRIV',E,ES,ED,ING)
VERBSTEM(ING) I N G --> VERB(SUP(VERBSTEM),PRESPART)
# I S --> VERB('BE',PRP3SG)

VERB --> VERBPHRASE(%VERB)

VERB('BE') VERBPHRASE(PRESPART) -->
                              VERBPHRASE(FORM=FORM(VERB),PROG)
```

Each of these rules can be "read" in a number of ways. For example, the first rule could be read:

> The string "#arriv" (where # means a space) is the stem of the verb "arrive", and it can take the endings "e", "es", "ed", and "ing".

or

> The string "#arriv" can be described by a VERBSTEM record that has a SUP attribute pointing to the record for "arrive" and has the indicators E, ES, ED, and ING. ('ARRIV' is the name of the record for "arrive".)

or

> If the string "#arriv" appears in the input stream, create a VERBSTEM record which has a SUP attribute of 'ARRIV' and the indicators E, ES, ED, and ING, to describe the string.

The first "reading" above gives part of the information that would be found with the entry for "arrive" in an English dictionary, and the second one is the sort of description given earlier in this section. Both of these are declarative statements. The third one is the way these rules are intended to be read, i.e. as imperative statements, because they are commands to the computer.

The second rule says that if a string described by a VERBSTEM record with an ING indicator is followed by the

string "ing", create a VERB record which has the same SUP attribute as the VERBSTEM and which has a PRESPART indicator, to describe the whole string. The third rule says that if the string "#is" appears, create a VERB record which has a SUP of 'BE' and a PRP3SG indicator to describe it. The fourth rule says that if there is a string described as a VERB, create an additional record to describe that string as a VERBPHRASE and have this new record be a copy (%) of the VERB record. The last rule says that if a string described by a VERB record with a SUP of 'BE' is followed by a string described as a VERBPHRASE record with a PRESPART indicator, create a new VERBPHRASE record which has all of the same information as the old VERBPHRASE record but has the FORM of the VERB (e.g. PRP3SG) and has a PROG indicator. Stated declaratively this rule says that any form of the verb "be" followed by a present participle forms the progressive.

Figure 1.7 shows the records which would be created by applying these rules to the string "#is#arriving". In each box the value of the syntactic-type attribute is given on the first line, the value of the SUP attribute is given on the second line, and the names of those indicators which are "on" are given on the third line. It can be seen that the third rule would be applied to the substring "#is", and the first rule would be applied to the substring "#arriv". Then the second and fourth rules would be applied to cover the substring "#arriving". Finally, the last rule would be applied, resulting in a VERBPHRASE record for the entire string with

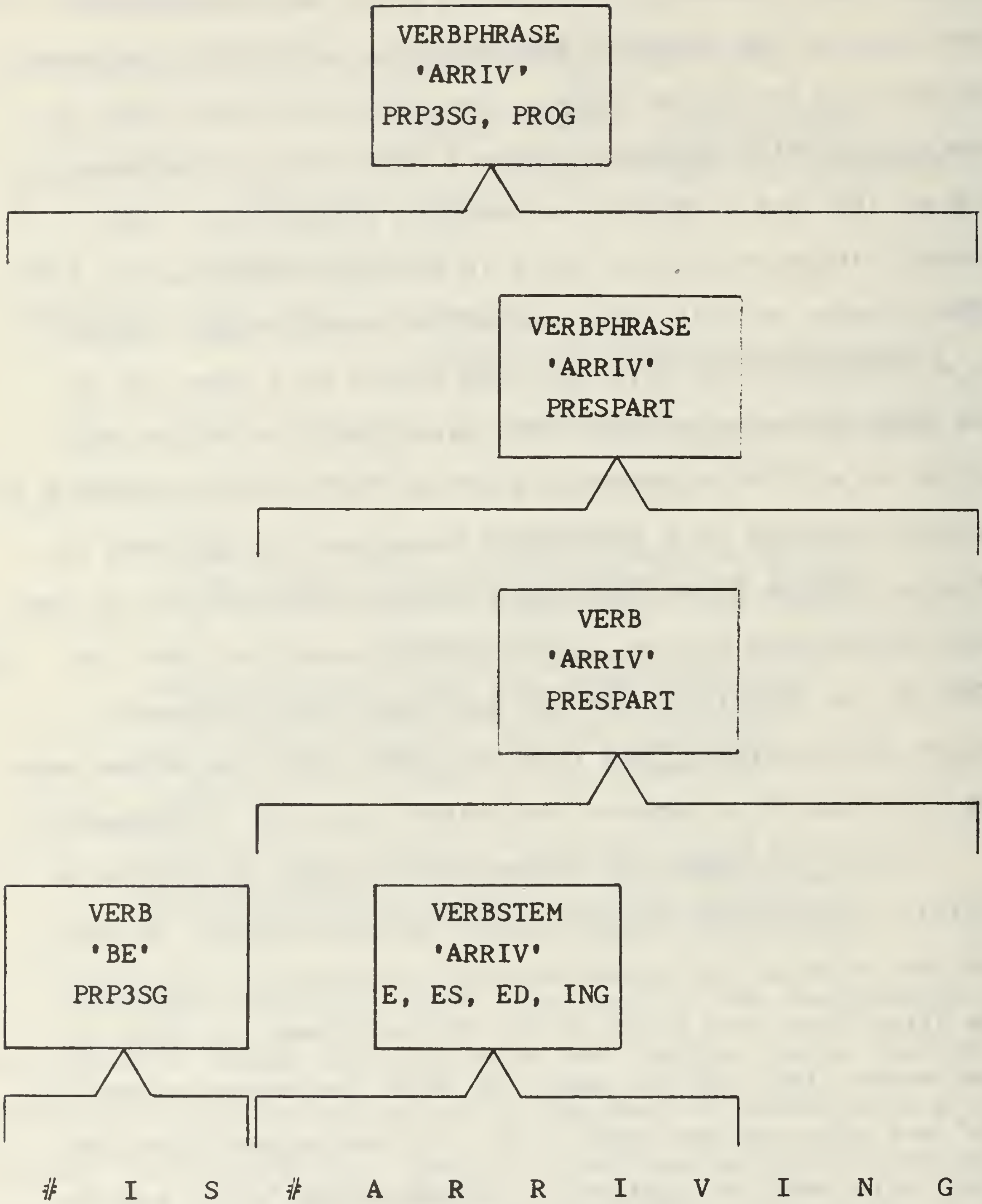


Figure 1.7. Records Created During Decoding and Encoding of the String "is arriving".

a SUP of 'ARRIV' and with the indicators PRP3SG and PROG.

2(b). Syntactic Structure

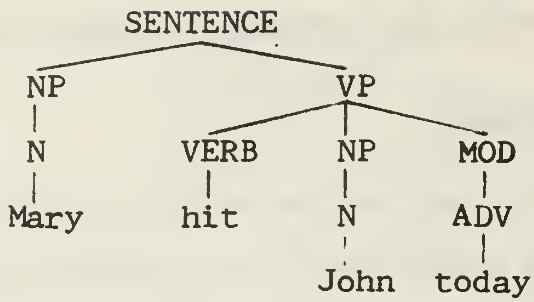
The parentheses on the left side of a rule are said to contain a condition specification and those on the right contain a creation specification. From these specifications it is possible to access essentially any information in the computer.

If these specifications were removed from the rules given here, what would be left are "phrase structure rules" of the sort that have been used by linguists for a number of years to describe the syntactic structure of sentences. However, the inclusion of these specifications makes decoding and encoding rules significantly different from phrase structure rules. In decoding, for instance, rather than just building a syntactic description of a sentence (i.e. a "parse tree"), any sort of description desired can be constructed in the form of records, with the exact manner of construction given by creation specifications.

A basic premise of NLPQ is that a "semantic" description is much more useful than a syntactic one. For example, it is felt that the sentence "Mary hit John today." would be more usefully described by the record

SUP	'HIT'
AGENT	'MARY'
GOAL	'JOHN'
TIME	'TODAY'

than by the tree



Having information organized semantically in record form can be helpful during processing also, because this information may be queried from condition specifications. For example, classifying phrases semantically as soon as possible helps to avoid some forms of syntactic ambiguity. In the sentence

"Ships are unloaded at a dock in the port."

a strictly syntactic analysis could not determine whether the second prepositional phrase "in the port" modifies the verb "unloaded" or the noun "dock." However, by considering that an action like unload takes place at one location and both "at the dock" and "in the port" are location phrases, then they cannot both modify the verb in this sentence. This results in the analysis that "in the port" modifies "dock" (i.e. specifies its location) and the whole phrase "at a dock in the port" modifies "unloaded" (i.e. specifies that the location of the action is a dock which is in the port).

The basic philosophy underlying all of this is that the analysis of text should not be done "in a vacuum", but rather as much information as possible should be made available and used to aid the analysis.

2(c). Encoding Rules

The following are examples of encoding rules:

```
VERBPHRASE(PROG)  -->  VERB('BE',FORM=FORM(VERBPHRASE))
                       VERBPHRASE(-PROG,-FORM,PRESPART)
VERBPHRASE  -->  VERB(%VERBPHRASE)
VERB('BE',PRP3SG)  -->  # I S
VERB(PRESPART)  -->  VERBSTEM(SUP(VERB))  I  N  G
VERBSTEM('ARRIV')  -->  # A R R I V
```

Each of these rules can be read declaratively, too, in a manner similar to the decoding rules. For example, the first rule says that the progressive is formed by having some form of the verb "be" followed by a present participle. However, these rules also are intended to be read as imperative statements, because they too are commands to the computer. The first rule, then, says that when encoding, if there is a VERBPHRASE record which has a PROG indicator, split it into a series of two new records. The first of these is a VERB record with a SUP of 'BE' and the FORM of the VERBPHRASE, and the second is a new VERBPHRASE record which has all of the same information as the old VERBPHRASE record but does not have the PROG indicator or the old FORM and instead has a PRESPART indicator. Then apply an appropriate encoding rule to each of these records.

The second rule would be applied to a VERBPHRASE record which does not satisfy the condition of the first rule, i.e. does not have a PROG indicator, and it says to replace the VERBPHRASE record by a VERB record which is a copy of it.

The third rule says that if there is a VERB record with a SUP of 'BE' and with a PRP3SG indicator, put the string "#is" into the output stream. The fourth rule says that if there is a VERB record with a PRESPART indicator, replace it by a VERBSTEM record with the same SUP, to be followed by the string "ing". The last rule says that if there is a VERBSTEM record with a SUP of 'ARRIV', put the string "#arriv" into the output stream.

Figure 1.7 also serves to illustrate the application of these encoding rules. Beginning with a VERBPHRASE record with a SUP of 'ARRIV' and with the indicators PRP3SG and PROG, the first rule would be applied, yielding a VERB record and a VERBPHRASE record. Then the third rule would be applied to the VERB record to produce the string "#is", and the second rule would be applied to the new VERBPHRASE record, yielding another VERB. Then the fourth rule would be applied to this new VERB, yielding a VERBSTEM and the string "ing", and finally the last rule would be applied to the VERBSTEM to produce the string "#arriv". The net result of all this is that the string "#is#arriving" would be put into the output stream. The records created during the encoding of this phrase are the same as those created during the decoding of the same phrase, except that the VERBSTEM record for 'ARRIV' would not have any indicators.

2(d). <u>Named Records</u>

It was stated earlier that information about words and concepts, i.e. the "long-term memory", is initially entered into the system by means of named record definitions. The following is a typical named record definition:

ARRIV ('EVENT',E,ES,ED,ING)

This defines a record named 'ARRIV' which has a SUP attribute pointing to the named record 'EVENT' and has the indicators E, ES, ED, and ING. These indicators provide information about the word "arrive", and the SUP attribute provides information about the concept "arrive" in this case.

It may be noted that what appears in parentheses in a named record definition looks similar to what appears in parentheses in a rule. Actually, the parentheses of a named record definition contain a creation specification of exactly the same sort as in a rule, and these specifications are compiled and executed by the same FORTRAN routines.

Named records are helpful during both decoding and encoding. For example, many of them can be treated as lexicon entries to eliminate the need for most word-building rules, such as the first and third decoding rules and the last encoding rule in the examples just given.

It should be pointed out that virtually none of the names used in rules and named record definitions have any predefined meaning in this system. The user is free to devise meaningful

names to suit himself just as a computer programmer devises symbolic names. He does have to tell the system which of the names he uses are indicator names, however.

3. Stratificational Linguistics

This research has been done within the framework of stratificational linguistics [18,21,23]. In this theory, language is considered to be a system of relationships existing in the brain for translating information in the form of text (either spoken or written), which is one-dimensional, into equivalent information in the form of a multi-dimensional network in the mind of the receiver, and vice versa. The first of these two processes (i.e. text-to-network) is called decoding, and the inverse process (i.e. network-to-text) is called encoding.

The NLP system is to the computer what the language system is to the brain in this theory. An attribute-value data structure is a multi-dimensional network, and NLP (the FORTRAN program, plus sets of decoding and encoding rules) is a system of relationships existing in the computer for translating information in the form of text into equivalent information in the form of an attribute-value data structure in the memory of the computer, and vice versa. These processes also are called decoding and encoding.

The feature of stratificational linguistics from which it got its name is that a language is considered to consist of several levels (strata), each of which can be described

separately, but in a similar fashion. The number of levels postulated has varied from time-to-time, so in NLP no assumption is made about the number of levels. Rules may be grouped in any manner the rule-writer desires.

In the rules written for NLPQ a three-level system is considered. The "morphology" deals with the manner in which characters are put together to form parts of words and parts of words are put together to form words. The "lexology" deals with the way in which words form phrases, phrases form clauses, and clauses form sentences. And, the "semology" is concerned with the relationship between the information in a sentence and the information in some particular portion of the Internal Problem Description.

A simple example will serve to illustrate the points discussed above. Either of the following two sentences could appear in an English description of a queuing problem:

> The men unload the ship at a dock for 8 hours.
> The ship is unloaded for 8 hours at a dock by the men.

At the morphological level the only really significant difference between the two is the form of the verb "unload". Adding "ed" to a verb stem to form the past participle is considered to be a morphological process. Also, there are two additional words in the second sentence ("is" and "by").

At the lexological level the two sentences are quite different, however. Each has a different subject ("men" vs. "ship") and a different ordering of the modifying phrases.

A typical process in the lexology is the one which puts a form of "be" together with a past participle to form a passive verb phrase (e.g. "is unloaded"). Other processes at this level would do such things as recognize what each prepositional phrase is for (i.e. location, duration, etc.).

At the semological level the two sentences given above are identical. They have exactly the same meaning and, therefore, would be related to exactly the same structure in the IPD. During decoding, after the completion of lexological processing, either of these sentences would be represented by the record:

SUP	unload
AGENT	men
GOAL	ship
LOCATION	dock
DURATION	8 hours

It would then be the task of the semological processing to merge this information properly into the IPD.

This brief introduction to stratificational linguistics should suffice at this point in the report. Even though this work has been done within the framework of this theory of language, the development here is not couched in the usual stratificational notation and terminology, but rather is done strictly in terms of attribute-value data structures and phrase-structure-like encoding and decoding rules. In the concluding chapter, however, there is a detailed discussion relating this work to stratificational theory.

C. THE COMPUTER SYSTEM

The computer system developed to meet the general objectives of this research (NLP) is in the form of a 5000-statement FORTRAN program which runs under the CP/CMS time-sharing system on an IBM 360/67. This program consists of about 100 routines and requires a virtual machine with at least 225K bytes of storage, not including the data area where rules and records are stored.

The main routine of the program serves as a monitor to provide for interaction with the user. There are several different "commands" and "heading lines" which it recognizes and which cause it to call appropriate subroutines. For example, the heading line NAMED RECORDS: causes it to call the routine for processing named record definitions, and the command PRINT 'ARRIV': causes it to call a routine which would print the named record 'ARRIV'.

There are several routines which form a "compiler" for the rule language. These routines are used to process decoding and encoding rules and named record definitions to convert them into their internal representation. Then there are "execution-time" routines which perform encoding and decoding according to this information. There is also a large group of subroutines which provide list-processing capabilities for the rest of the system.

The program has facilities for handling input and output in a general fashion so that either the terminal or card-image files can be used interchangeably. It is possible to

store the entire data area in binary form so that the rules do not have to be compiled at the start of each terminal session. This also makes it possible to suspend a terminal session at any time. The program also has extensive optional tracing capabilities which are especially helpful when "debugging" a set of rules.

To meet the specific objectives of this research (NLPQ), approximately 300 named record definitions and 800 encoding and decoding rules were written in the language of this system. These, preceded by relevant declarations, are listed in Appendices A through I of this report. (The numbers which appear in parentheses to the left of each rule in the listings were not included as input to the system but were added to the listings for purposes of reference from Chapter III and IV.) To compile all of these into their internal representation takes about 100 seconds of virtual CPU time (about 3 minutes total CPU time). This information, along with temporary storage needed during processing, requires an additional 100K bytes, on top of the 225K required by the FORTRAN program.

Further discussion about the program is included at appropriate points in Chapters II-IV. Also, detailed discussion about the material in the appendices is given in those chapters.

D. ORGANIZATION OF THIS REPORT

This introductory chapter was intended to give the reader an overview of what has been done, without delving into the technical details. First the capabilities of NLPQ, the

specific system for queuing problem simulation programming, were illustrated by means of a sample problem and discussion about its processing. Then an introduction to the information structure and rule language of NLP, the general system for natural language processing, was given. Finally, there was a brief description of the FORTRAN program which does the processing.

The next three chapters present the technical details. Each chapter consists of essentially two parts, the first of which is about the general system (NLP) and the second of which is about its use for the queuing problem application (NLPQ). Chapter II discusses the information structure used by this system, Chapter III discusses encoding and Chapter IV discusses decoding. Because decoding logically precedes encoding, it may seem that the topics of Chapters III and IV are in the wrong order. However, encoding is a more straightforward process than decoding and provides a simpler introduction to the rule language of this system. These three chapters are <u>not</u> independent of one another.

The final chapter (V) relates the work done here to that done by others and suggests directions for future research. This includes discussions of how this work is related to stratificational linguistics and how the system developed here compares with other natural language processing systems developed at about the same time.

II. THE INFORMATION STRUCTURE

In describing any computer process it is necessary to state both the structure of the information that is being processed and the procedure by which the information is processed. The information structure for the system under discussion is presented in this chapter of the report. The procedure used for processing this information is presented in the following two chapters on encoding and decoding.

This chapter begins with a general discussion of the manner in which information is stored in this system for any application, including implementation details. This is then followed by a discussion of that portion of this information which is about words and concepts for dealing with simple queuing problems. Finally, the manner of representing the information about any particular queuing problem (i.e. the Internal Problem Description) is discussed in detail.

A. GENERAL DISCUSSION

Almost all information in this system is maintained in the entity-attribute-value form. This form of information structure has been used extensively in previous artificial intelligence work, and also in simulation modeling. Most notably the "world views" of both the SIMSCRIPT and the GPSS simulation programming systems consider the world to be composed of entities, each of which can be described by giving the values of its attributes. For example, in describing a man, it might be said that he has a height of 6 feet and a weight of 165 pounds; where height and weight are attributes,

and 6 feet and 165 pounds are their values. This manner of description can be used for abstract entities also. For example, an action may be described by specifying its agent (i.e. the performer of the action), the location where the action takes place, the time when the action takes place, etc.

This section presents the details about the manner in which the entity-attribute-value data structure concept has been implemented for this system on the IBM 360, and how particular structures may be established. The two sections following this one describe particular structures.

1. The Cell

The basic unit of information in this system is the cell. With two minor exceptions each cell is capable of holding the value of one attribute. A number of cells may be strung together (i.e. linked) to form a list of attribute values. Such a list is called a record, and is the computer's counterpart of the actual entity being described.

In the 360 computer each cell consists of 8 contiguous bytes of core storage. These 8 bytes are considered to be grouped into four fields of two bytes each. The leftmost field is called the type (TYPE); the next field is called the attribute (ATTR); the third field is called the address (ADDR); and the last field is called the link (LINK). The ATTR field holds a number in the range 0 to 32767 which is considered to be the attribute number. For example, the number 759 might correspond to the attribute "height". The way in which these numbers are assigned will be discussed shortly. The LINK

field contains the identification number of the next cell on the list, with a LINK of zero indicating that this is the last cell on the list. Each cell has a unique identification number which is in the range 1 to 32767, and actually corresponds to the subscript by which this cell is accessed in the FORTRAN program. The ADDR field contains either the actual value of the attribute or points to another cell which contains the value, depending on the TYPE field. The TYPE field may be either 0, 1, 2, or 3, to designate the type of attribute value.

A type-0 cell has a two-byte value which is stored in the ADDR part of the cell. A type-1 cell has an eight-byte value which is stored in a cell which is pointed to by the ADDR part of this cell. This is used primarily for holding alphanumeric data, such as names. The ADDR part of a type-2 cell points to a "local list" of cells. This is called a <u>local</u> list because it is not pointed to by any other cell in the structure. A local list is normally used to hold multiple values of an attribute, but also has other uses which will be mentioned later. In a type-3 cell the ADDR part points to another record. The four types of cells are illustrated in Figure 2.1.

2. <u>The Reference Counter</u>

The cells of a record are linked in order by attribute number. The first cell of each record is a special attribute called the "reference counter". It is attribute number 0, and it is a type-0 cell, whose value indicates how many type-3 cells elsewhere in the information structure point to this particular record. This reference counter idea has been used

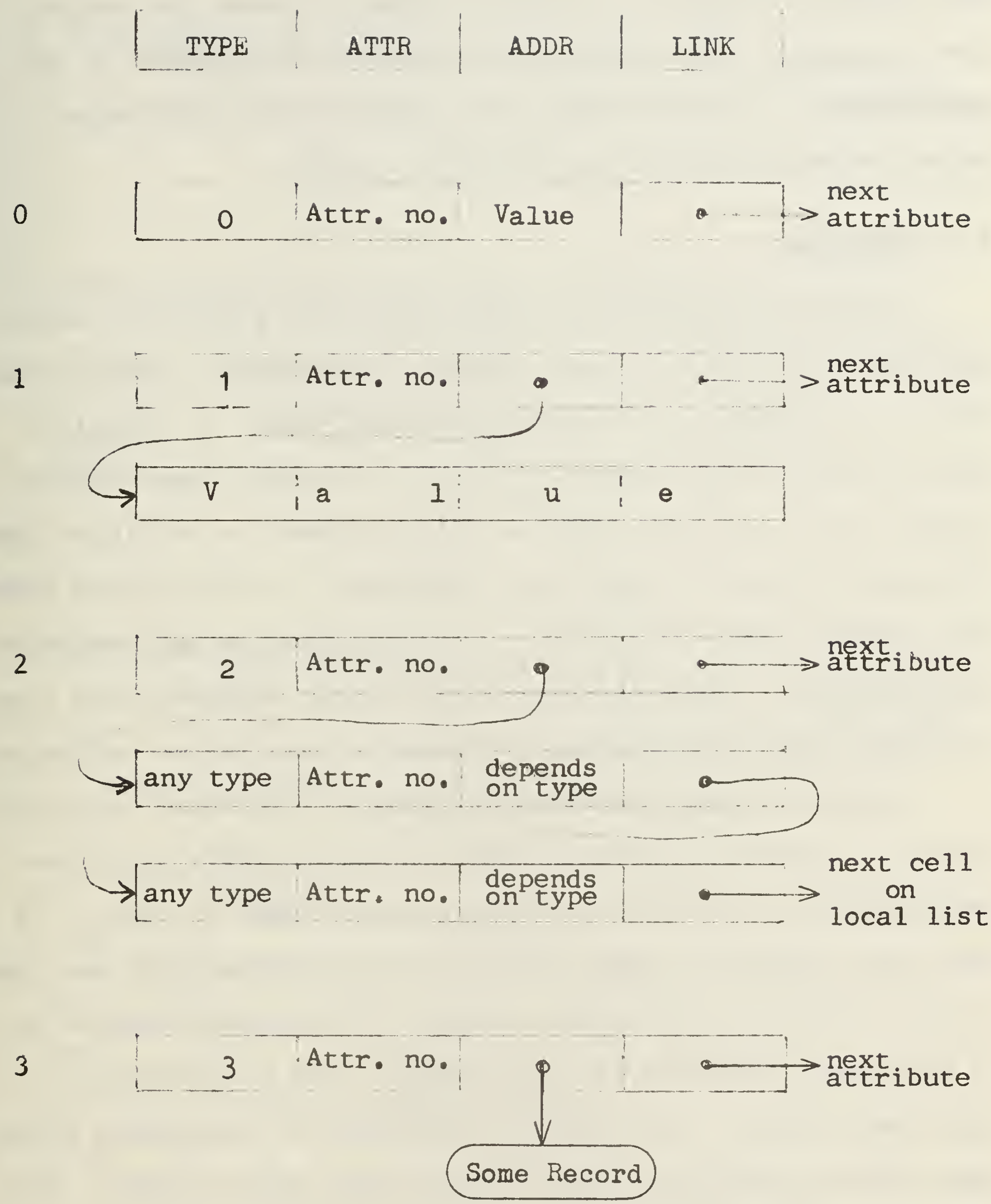


Figure 2.1. The Four Types of Cells

previously in the SLIP list processing system [41]. When the reference counter of a record drops to zero, the record can be erased. The identification number of a record is the same number as the identification number of the cell which holds the reference counter for that record.

3. Indicators

In describing an entity there are often attributes which can take on only one of two values. For example, the attribute "sex" of a person may take on the values "male" or "female", which could be represented by 0 or 1. It would seem wasteful to use a full cell to store the value of such an attribute when all that is needed is one bit. Therefore, in this system there is a special kind of attribute called an "indicator", which is binary valued. These indicators are simply numbered from 1 up to however many are needed. For example, sex may be indicator 7.

Indicators are stored collectively as the value of an attribute, currently number 2, with each bit in this value corresponding to an indicator (numbered from right to left). If there are fifteen or fewer indicators being used, then one type-0 cell can hold all of their values. If there are from 16 to 55 indicators being used, then a type-1 cell must be used to hold their values. If there are more than 55 indicators being used, then a type-2 cell with a local list must be used. This manner of storing indicator values is illustrated in Figure 2.2. As will be seen in later sections of this report, indicators are particularly useful for holding information about word forms, such as "singular", "plural", "past participle", "present tense", "perfect" etc.

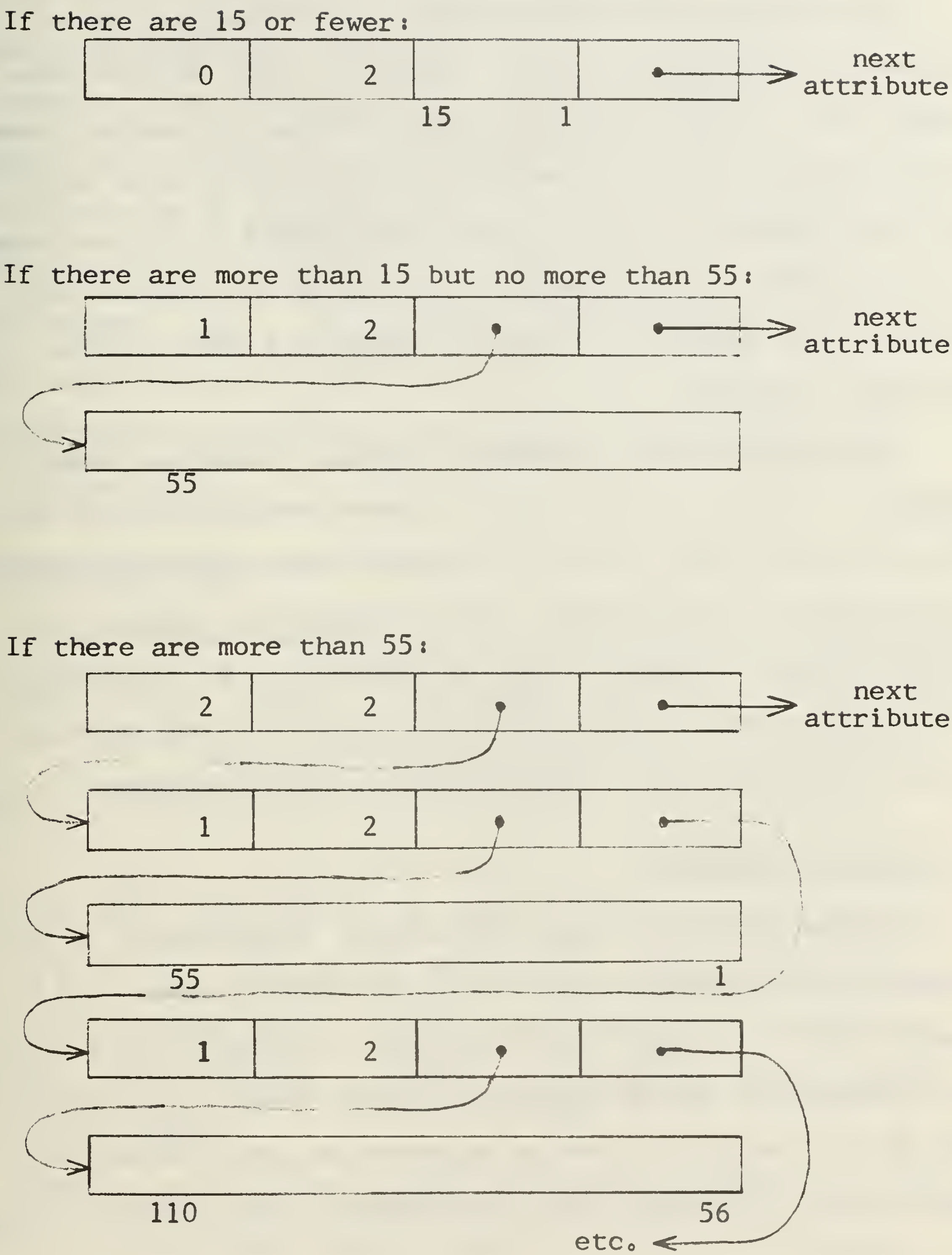


Figure 2.2. The Manner in which Indicators Are Stored

4. Named Records

An attribute which has special significance in this system is the NAME attribute. Currently, this is attribute number 10. It is a type-1 cell which has as its value a name consisting of eight or fewer characters, the first of which must be alphabetic.* This name is considered to be the "name" of the record. These names are made up by the user of this system in the same way as variable names are made up when writing a program.

A record that has a NAME attribute is called a "named record." As will be seen shortly, named records furnish the means by which most static information about words and concepts is entered into the system. The most important feature of named records is that they may be referred to by name both from other named records and from the encoding and decoding rules, as will be seen in later sections of this report.

5. The SUP Attribute

Another attribute which has special significance in this system is the SUP attribute. It is attribute number 1 of a record, and is a type-3 cell pointing to another record which is considered to be the superset of this record. For example, the SUP attribute of a record representing the concept "car" might point to another record representing the concept "vehicle".

---

*Actually, throughout this system longer names may be used, but they must be unique in the first eight characters.

Furthermore, the SUP of vehicle may be "mobile entity"; and the SUP of mobile entity may be "entity". A series of SUP's of this sort is referred to as a "SUP chain". The idea of a SUP chain is basic to the functioning of "semantic event forms" in the PROTOSYNTHEX system [35]. It is also tied in with the idea of "taxonomic hierarchy" which is discussed in some of the literature on Stratificational Grammar [20]. SUP chains are very important in the system under discussion here, and facilities for effectively utlizing them are included in the grammar rule language, as will be seen later.

6. <u>Attribute Names</u>

As has already been pointed out, in this system attributes are identified by numbers in the range 0 to 32767. The numbers 0, 1, 2, and 10 have already been assigned for the reference counter, SUP, indicators, and NAME attribute, respectively, for the kinds of records being discussed in this chapter. For any particular application the user is free to assign the numbers from 11 on in any manner he desires. In writing grammar rules and defining named records he may refer to an attribute by using an @ followed by a number. For example, he may write @1, @15, or @1764.

However, it is usually more convenient to refer to an attribute by name rather than by number, so there are facilities in the system for associating a name of up to eight characters with an attribute number. There are actually two

facilities for doing this, the first of which may be considered to be explicit, and the other implicit. In order to explicitly associate names with attribute numbers, it is necessary to include in the deck of inputs to the system a card with the word ATTRIBUTES in columns 1-10 and a colon in column 11, followed by one or more cards of name-number pairs. For example, the following may be given:

```
ATTRIBUTES:
      SUP  1,  NAME  10,  WEIGHT  37
```

After this declaration is made, attribute 1 may be referred to as SUP, attribute 10 as NAME, and attribute 37 as WEIGHT. Exactly where these names are used will be seen shortly. Actual attribute name declarations used may be seen in the listing in Appendix A.

Implicit association of a name and an attribute number comes about simply by using in a grammar rule or in a named record definition a name which has not been declared in any way (except possibly as the name of a named record). The first time the name is encountered by the system, a named record with that name is constructed, and from then on <u>the identification number of that named record will be the attribute number associated with that name</u>. For example, if the named record with the name HEIGHT (i.e. @10="HEIGHT") had an identification number of 759 (i.e. the reference counter of this record is stored in cell 759), then the value of the HEIGHT attribute of some other record would be stored as attribute

759 of this other record (i.e. the value would be stored in a cell whose ATTR field had the number 759 in it). There is no need for the user to ever be aware of the actual number associated with a name in this manner because he would always refer to the attribute by name. It is worth noting, however, that all numbers assigned in this way will be greater than 300, so there is no problem of interference with numbers explicitly assigned (as long as they are less than or equal to 300).

7. Indicator Names

All indicators to be used in a particular application of this system must be explicitly given names. It is possible for an indicator to have more than one name, and it is also possible for a name to "cover" more than one indicator. In addition, a name may be declared to be associated with one indicator or group of indicators in a named record and a different one or group in any other kind of record. The manner in which indicators are given names is similar to the manner in which attributes are explicitly given names. For example, the following cards could be included in the input deck:

```
INDICATORS:
          PRESENT  5,   PAST  6,   TENSE  5-6,
          SING  7  '4',   PLUR  8  '5',
          NUMB  7-8  '4-5',   PERS  9-11,
          PERS1  9,   PERS2  10,   PERS3  11
          VERBFORM  5-11
```

After this declaration is made, if the name PRESENT appeared in a grammar rule or in a named record definition, it would be taken to refer to indicator number 5 (i.e. the fifth bit from the right in the value of attribute 2). The name TENSE would refer collectively to indicators 5 and 6. The name SING would refer to indicator 7, except in a named record, where it would refer to indicator 4. VERBFORM would refer collectively to all of the indicators from 5 to 11. When a name is not given a number in quotes, the number which it is given is associated with it both in a named record and in any other kind of record. Actual indicator name declarations used may also be seen in the listing in Appendix A.

8. Defining Named Records

Named records may be defined (i.e. given initial attribute values) by entering cards that have the names, followed by the attribute values in parentheses. For example, the following may be given:

```
NAMED RECORDS:
          WILL  (FUTURE,PS='VERB',SING,PLUR)
          CARGO  ('ENTITY',ES)
          ENTITY  (PS='NOUNSTEM')
          MISC  (COUNT(MEMORY)=3,
                     @15(PS('ENTITY'))="ABCDEFGHIJ")
```

If FUTURE, SING, PLUR, and ES had previously been declared as indicator names, then the above six cards would explicitly define four named records ('WILL', 'CARGO', 'ENTITY', and

'MISC') and could also result in the creation of four others ('PS', 'VERB', 'NOUNSTEM', and 'COUNT') depending upon what other information had preceded this.

The named record 'WILL' would have three indicators set and have a PS attribute pointing to the named record 'VERB'. This may be telling the system that the word "will" is both the singular and plural form of a verb which when used as an auxiliary verb helps to form the future tense. The named record 'CARGO' would have a SUP attribute pointing to 'ENTITY' (i.e. the named record 'ENTITY') and have the ES indicator set. This may mean that the concept "cargo" is in the class of physical entities and that the plural of the word "cargo" is formed by adding "es". 'ENTITY' would have a PS attribute of 'NOUNSTEM' (i.e. pointing to 'NOUNSTEM'), which may be used to tell the system that the names of physical entities are nouns. PS stands for part-of-speech.

The final definition in the above example was included just to illustrate a couple of features of the definition language. The named record 'MISC' would have no attributes other than a reference counter and a NAME, which all named records have. The attributes being given values within those parentheses are actually in other records. The first specification element sets the COUNT attribute of the MEMORY record to the numeric value 3. MEMORY is a special record which is usually used to hold global information. It may also be referred to as MEM. The second element sets attribute 15 of the record pointed to by the PS attribute of 'ENTITY' to the

character string value ABCDEFGHIJ. The left side of the equality could have been written as @15('NOUNSTEM'), because PS of 'ENTITY' is a pointer to 'NOUNSTEM'.

Whenever a name appears in single quotes, it is considered to be the name of a named record. If it is the first time the name is encountered, a record is created for it, with a NAME attribute and a reference counter, of course. That is how the records for 'VERB', 'ENTITY', and 'NOUNSTEM' would come into being in the above example. The definition given there for 'ENTITY' would merely add the PS attribute to the record already created.

Whenever a name which has not been previously declared (e.g. as an indicator name) is encountered not in quotes, it is considered to be an attribute name. This is the "implicit association" which was discussed in the section on attribute names. If there were not already a named record by this name, one would be created at this time. In the above example the records for 'PS' and 'COUNT' would come about in this manner. Then the identification number of the 'PS' record would be used as the attribute number for the PS attribute in the 'WILL' and 'ENTITY' records, and the identification number of 'COUNT' would be the attribute number of the COUNT attribute of MEMORY.

From the preceding example, the manner in which attribute values are specified can be seen. The whole of what appears in parentheses after the name of the record is called a "creation specification". It consists of one or more "specification elements", separated from one another by commas.

Creation specifications appear in encoding and decoding rules in this system, also, and a detailed discussion of all of the various kinds of specification elements which can be used will be included in the next chapter of this report, as part of the explanation of encoding rules. For the current chapter it is sufficient to understand the following three types of specification elements:

> <u>indicator name</u> -- Set that indicator.
>
> <u>named record name</u> (in quotes) -- Have the SUP attribute point to that named record.
>
> <u>attribute = value</u> -- Give that attribute the value.

In the last element above, the attribute may be simply an explicit or implicit name or a number (with an @), or it may be the more complicated form of "attribute of attribute of attribute ... of record", where each left parenthesis may be read as "of". The value may be a number (which results in a type-0 cell), or a character string in double quotes (which results in either a type-1 cell or a type-2 cell with a series of type-1 cells), or a record name (which results in a type-3 cell).

Actual named record definitions used may be seen in the listings in Appendix B.

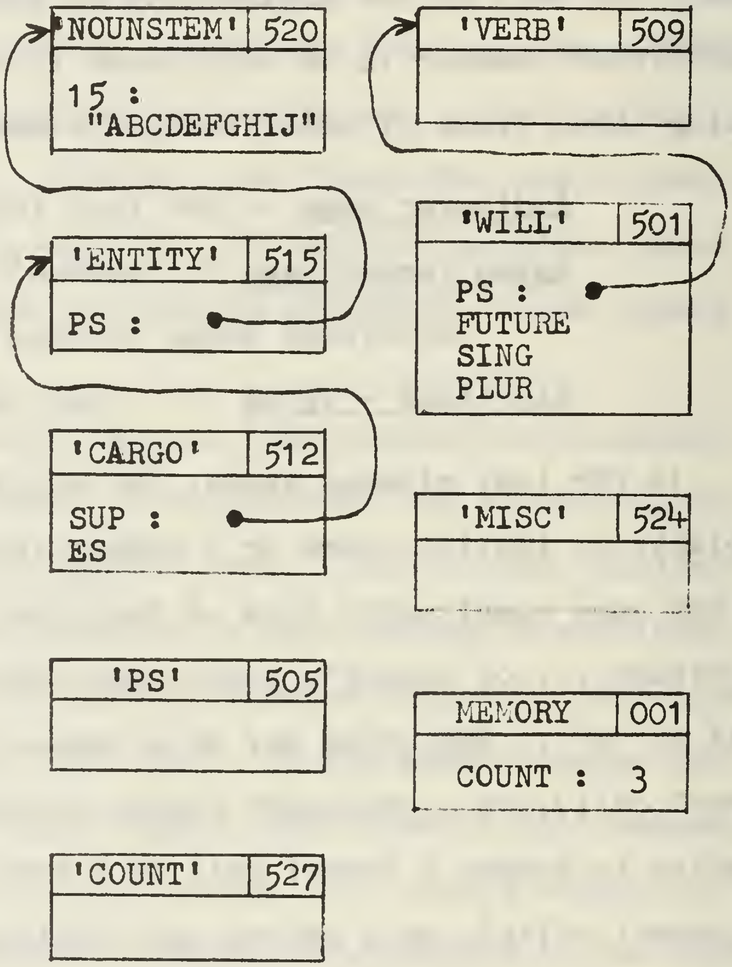
9. <u>An example of Cell and Record Structure</u>

Figure 2.3 furnishes an illustration of all of the material discussed so far in this chapter. At the top of the figure are seven cards that could be input to the system. Beneath

```
INDICATORS:
    FUTURE  1,  SING  2,  PLUR  3,  ES  4

NAMED RECORDS:
    WILL  (FUTURE,PS='VERB',SING,PLUR)
    CARGO  ('ENTITY',ES)
    ENTITY  (PS='NOUNSTEM')
    MISC  (COUNT(MEMORY)=3,@15(PS('ENTITY'))="ABCDEFGHIJ")
```

Cell No.	TYPE	ATTR	ADDR	LINK
001	0	0	1	530
501	0	0	1	504
502	1	10	503	508
503	W I L L			
504	0	2	7	502
505	0	0	1	506
506	1	10	507	0
507	P S			
508	3	505	509	0
509	0	0	2	510
510	1	10	511	0
511	V E R B			
512	0	0	1	518
513	1	10	514	0
514	C A R G O			
515	0	0	2	516
516	1	10	517	523
517	E N T I T Y			
518	3	1	515	519
519	0	2	8	513
520	0	0	2	521
521	1	10	522	531
522	N O U N S T E M			
523	3	505	520	0
524	0	0	1	525
525	1	10	526	0
526	M I S C			
527	0	0	1	528
528	1	10	529	0
529	C O U N T			
530	0	527	3	0
531	2	15	532	0
532	1	15	533	534
533	A B C D E F G H			
534	1	15	535	0
535	I J			



Top: Input to the system

Left: Internal representation (i.e. cell structure)

Right: Graphic representation of record structure

Figure 2.3. An Example of Cell and Record Structure

that, on the left is the actual cell structure that would result from the processing of the named records, and on the right is a graphic representation which shows the record structure more clearly. It should be noted that the three parts of the figure are just three different representations of the same information.

In the graphic representation each record is shown as a box with three parts. The name and the identification number (i.e. the number of the reference counter cell) appear at the top of the box, and the attribute values appear in the bottom part. Two attributes -- the reference counter and NAME -- are not included in the bottom part of the box, because the value of the reference counter can always be determined by noting how many arrows point to the record, and the value of NAME is just the alphanumeric representation of the name of the record given in the top part of the box. An attribute and its value appear separated by a colon. If the value is a pointer to another record, the pointer is actually drawn. Indicators which are set (i.e. have a value of 1) are simply listed after the other attributes.

The MEMORY record always begins in cell number 1. However, the selection of cell 501 as the starting point for the named records was arbitrary for this example. The cells shown are essentially what would be produced by the computer for this example, but some cells which would be part of "index lists" have been left out for simplicity. (Each named record is pointed to from an index list of records whose names begin with the same letter.) The values of all reference counters have

been increased by 1 also, to reflect the fact that each record would be pointed to from an index list. The order in which the cells appear in the figure is exactly the order in which they would be created by the system. However, most LINK fields would get changed a number of times as a record is being constructed.

Any record can be traced through the cell structure if its starting point is known. For example, 'CARGO' begins at cell 512, and consists of cells 512, 518, 519, and 513. Actually, cell 514, which contains the full-cell value for cell 513, could be considered to be part of the record because it would not be pointed to from any other cell in the structure. (Although in this example all type-1 cells have their values in the very next cell, this would not always be the case.) The value of attribute 2 of this record (in cell 519) has the numeric value 8. In binary this is 1000, which means that only indicator 4 (ES) is set. Similarly, the value 7 (binary 111) in attribute 2 of record 501 (i.e. cell 504) means that indicators 1, 2, and 3 are set in that record.

The use of named record identification numbers as attribute numbers can be seen in the cell structure, also. For example, cell 523 holds the PS attribute (number 505) of 'ENTITY', and is a pointer to 'NOUNSTEM' (record 520). Cells 508 and 530 are similar. Also of interest is the manner in which the character string ABCDEFGHIJ is stored. Cell 531 is a type-2 cell, pointing to a local list of type-1 cells, each of which holds eight characters of the string.

The discussion in this section has been concerned with the way in which information about words and concepts can be entered and stored in this system <u>for any application</u>. In the next section the group of named records used to hold information about the words and concepts for simple queuing problems is discussed in detail. This is then followed by a section giving the details about the structure of information for any particular queuing problem (i.e. the Internal Problem Description.

B. INFORMATION ABOUT WORDS AND CONCEPTS FOR QUEUING PROBLEMS

Currently, there is a group of approximately 300 named records with information about words and concepts for simple queuing problems, sufficient to illustrate the system's capabilities. A listing of cards for defining these named records can be seen in Appendix B. (The declarations listed in Appendix A would be entered into the system first.) A more compact representation of essentially the same information is given in Figures 2.4, 2.5, and 2.6 in this section, in order to clarify the discussion of it. These figures furnish an outline for the discussion.

1. <u>Entities</u>

Two classes of physical entities may be readily distinguished in simple queuing systems. There are those that move through the system (e.g. cars in a gas station, customers in a bank, or ships in a harbor), and there are those that are a

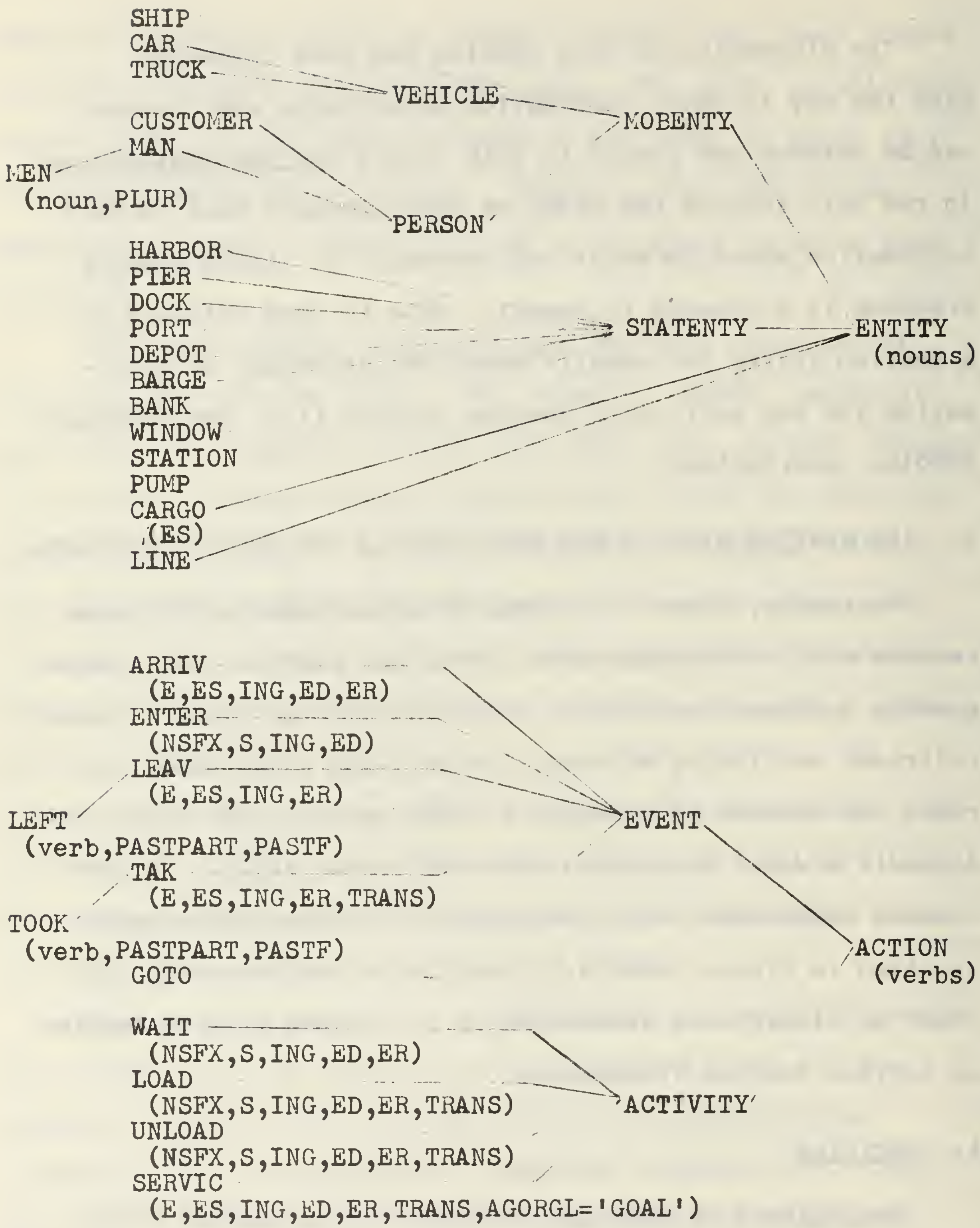


Figure 2.4. Named Records in the Sets ENTITY and ACTION, Showing the Superset Structure

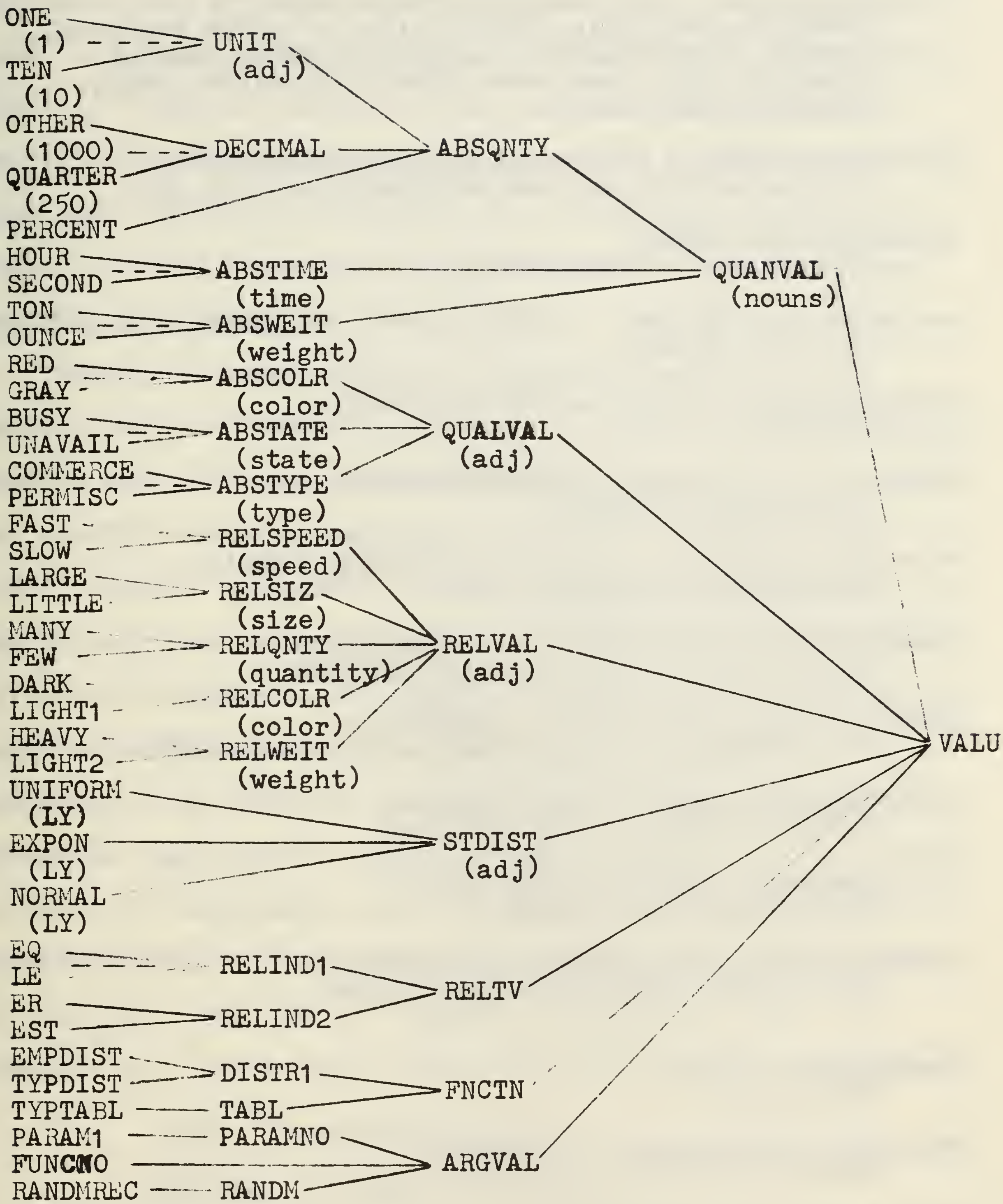


Figure 2.5. Named Records in the Set VALU, Showing the Superset Structure

Attribute names -- in the set ATTR(PS='NOUNS'):
PROBTIME,AGENT,GOAL,DURATION,IETM,LOCATION,PRED,SUCC,
CONDITN,QUANTITY,SIZE,COLOR,WEIGHT,SPEED,CONSUMP,
CAPACITY,OWNER,MEAN,RANGE,STDEV,TYPE,STATE,LENGTH,
RATE,TIME,IDNO,IDNAME

Attribute verbs -- in the set ATTRVERB(PS='VERBS'):
BE,IS,ARE,BECOM,HAV,HAS,TAKTIME,LOCAT,HOLD,HELD

Miscellaneous verbs:
WILL,DO,CAN, OCCUR,HAPPEN

Location descriptors -- LOCDESCR(PS='PREP',ATTRIB='LOCATION'):
AT,IN,ON,BY,NEAR

Determiners -- in the set DET(PS='ADJ'):
A,AN,THE

Fillers -- in the set FILLER(PS='ADV'):
JUST,SIMPLY,IMMEDIAT

Personal pronouns -- in the set PERS(PS='PRON'):
HE,SHE,HIM,HER,IT,THEY,THEM

Miscellaneous prepositions -- (PS='PREP'):
OF,FOR,FROM,TO,EVERY,PER,WITH,BETWEEN

Miscellaneous adjectives -- (PS='ADJ'):
THIS,THESE,LESS,EQUAL,GREATER,STANDARD,DISTRIBU

Miscellaneous nouns -- (PS='NOUNS'):
DEVIATIO,SIMULATIO,PROBLEM,PLACE

Miscellaneous adverbs -- (PS='ADV'):
THERE,OTHERWIS,NOT,THEN

Miscellaneous conjunctions -- (PS='CONJ'):
AND,IF,AFTER,WHEN,BEFORE,UNTIL,THAN

Conditions -- COND:
COND1,COND2

Successor descriptors -- SUCDESCR:
FRACTNL,PTYP,QTYP,FTYP,STYP

Record lists -- RECLIST:
ACTNLIST,MOBLIST,STALIST,DSTRLIST,SCSRLIST,MISCLIST,
UNITLIST

Figure 2.6. Other Named Records, Grouped by SUP or PS Attribute

permanent part of the system (e.g. gas pumps, tellers' windows, or docks). Here, the first of these are called "mobile entities", and the latter are called "stationary entities". Mobile entities correspond to transactions in GPSS and to temporary entities in SIMSCRIPT, and stationary entities correspond to facilities and storages in GPSS and to permanent entities in SIMSCRIPT.

There is a named record to represent the concept of physical entity ('ENTITY'), another for mobile entity('MOBENTY'), another for stationary entity ('STATENTY'), and several others for specific kinds of entities within these classes (e.g. 'SHIP', 'CAR', 'CUSTOMER', 'DOCK', 'WINDOW', and 'PUMP'). As has already been pointed out, the intended use of the SUP attribute is to express the superset relationship between objects represented by records. The upper part of Figure 2.4 shows graphically the superset structure for entities. The names appearing there are the names of named records, and the values of their SUP attributes are shown by the lines, which are pointing from left to right. For example, the SUP of 'TRUCK' is 'VEHICLE', the SUP of 'VEHICLE' is 'MOBENTY', etc. In Appendix B it can be seen that this information is entered as

```
TRUCK  ('VEHICLE')
VEHICLE  ('MOBENTY')   etc.
```

Most of the named records in the group being discussed in this section have only a SUP attribute defined. (Of course, they also have a reference counter and a NAME attribute.) However, some of them do have indicators and other attributes. In

the figure this additional information appears in parentheses beneath the name. Because of space limitations in the figure, the value of the PS attribute (part-of-speech) is simply given as a name in lower case letters. For example, "noun" means "PS='NOUN'", as can be seen in the appendix.

These records (with the exception of 'MEN', which will be discussed shortly) actually represent concepts, but they can also be used to hold information about the word which is the name of the concept. For example, the ES indicator in 'CARGO' says that the plural of "cargo" is "cargoes". (The plural of the others is formed simply by adding an "s".) This information is utilized in the English rules both for decoding and for encoding.

There are facilities available in the grammar rule language which make it convenient to use the information structured in the manner being described. For example, all of the records in the upper part of Figure 2.4 would satisfy the condition $'ENTITY', which means "in the set 'ENTITY'". Some of them would also satisfy the condition $'MOBENTY', and some others would satisfy the condition $'STATENTY'. Another facility makes it possible to obtain the value of an attribute from some record higher in the structure (i.e. to the right in the figure) without knowing how many levels away it is. For example, PS($('CAR')) or PS($('VEHICLE')) or PS($('CARGO')) would yield 'NOUNS'. ('NOUNS' stands for "noun stem". There are English decoding rules which will let a noun stem by itself or a noun stem followed by an "s" or "es", whichever is

appropriate according to the absence or presence of the ES indicator, become a noun.) However, PS($('MEN')) would yieldd 'NOUN', rather than 'NOUNS', because that is the first value of the PS attribute encountered in the SUP chain starting at 'MEN'. The use of these facilities will be fully described in the next two chapters of this report.

There are two more points about the upper part of Figure 2.4 which deserve mention. The named record 'MEN' does not represent a concept, but merely serves to give information about the irregular noun "men". Its SUP attribute is being used to point to the concept of which this is a name. This information is utilized by the English decoding rules. (Incidentally, the current grammar would also accept "mans" as the plural of "man".) The named records 'CARGO' and 'LINE' are not in either of the sets 'MOBENTY' or 'STATENTY'. This is because lines have special significance in queuing problems and therefore are given somewhat special treatment, and in the problems considered so far cargo has been rather insignificant, and therefore has been treated specially.

2. Actions

Generally speaking, in queuing systems mobile entities engage in "actions" at stationary entities. For example, ships arrive at a harbor, and cars are serviced at a pump. These actions may conveniently be separated into two groups: those that are instantaneous (e.g. arrive, enter, and leave), and those that consume time (e.g. service, load, and wait). Here

the first of these are called "events" and the latter are called "activities". In the GPSS language there are some blocks which can represent events (e.g. GENERATE and TERMINATE) and others which can represent activities (e.g. ADVANCE and GATE). In a SIMSCRIPT program there must be a routine written for each event, and each activity must be viewed as two events --the start of the activity and the end of the activity.

The lower part of Figure 2.4 shows graphically the superset structure for actions in the same manner as was already discussed for entities. However, it may be noted that these named records contain a lot more information about words than the records in the upper part of the figure do. This is so because action words are verbs, and verbs take on many more forms than nouns do in the English language. Each action name is considered to be a "verb stem" (as can be seen by the fact that PS('ACTION')='VERBS'), and some of the indicators are used by the decoding and encoding rules to form a verb from a verb stem and an appropriate suffix. For example, the E indicator in 'ARRIV' would be used by a rule which would recognize "arrive" as either the present plural or the infinitive form of a verb. The NSFX indicator in 'ENTER' (where NSFX stands for "no suffix") would allow "enter" to be recognized similarly. The meanings of S, ES, ING, and ED should be obvious at this point. The ER indicator could be used to form a noun (the so-called "agentive"), but it is not currently being utilized by the system.

It was stated earlier that a named record may contain

information both about a concept and about the word which is the name of the concept. However, the term "word" is really not defined well enough to be entirely useful here. (e.g. Is "arrive" a different word than "arrives"?) In Ref. [19] Lamb gives an excellent discussion in favor of using terms such as "morpheme", "lexeme", and "sememe" to be more precise. These are the names of the basic units of the "morphological", "lexological", and "semological" strata in a Stratificational Grammar description of a language. ("Arrive" and "arrives" are then two different morphological words, but they are two forms of the same lexical word.)

Using this terminology, the name of some named records in this system may be considered to be the name of a morpheme, the name of a lexeme, <u>and</u> the name of a sememe; and the record may contain information about all three. For example, the E, ES, ING, and ED indicators of 'SERVIC' specify to the rules that the <u>morpheme</u> 'SERVIC' can take any of those four suffixes to form the lexeme 'SERVIC'. The TRANS indicator specifies that the <u>lexeme</u> 'SERVIC' is transitive (i.e. can take a direct object in an active sentence), regardless of which form it is (i.e. "service", "services", "servicing", or "serviced"). The SUP attribute gives information about the <u>sememe</u> 'SERVIC', i.e. it says something about what 'SERVIC' means. The AGORGL (agent-or-goal) attribute also gives semological information. It specifies that the mobile entity involved in a service action is normally the "goal" of the action. i.e. the one being serviced. In the other actions

shown in the figure, the mobile entity is normally the "agent", i.e. the one performing the action. Actually, for 'LOAD' and 'UNLOAD' it could be considered to be either, depending on the way the statement is made (e.g. "Ships load cargo." vs. "Ships are loaded.").

The named records 'LEFT' and 'TOOK' serve a purpose similar to 'MEN', which has already been discussed. They give information about irregular verb forms. For example, 'LEFT' says that "left" is the past participle and the simple past form of the <u>lexeme</u> 'LEAV'. (In the English grammar rules developed for this system noun stems and verb stems are considered to be morphemes, and nouns and verbs are lexemes.)

It should be noted that if the terms "event" or "activity" or "action" were to appear in English text being decoded by this system with the SUP structure shown, they would be taken to be verb stems. However, because their named records have no suffix indicators, the decoding rules would not allow them to achieve verb status. If these terms were expected to appear in queuing problem statements, different names could have been chosen for these named records (e.g. 'EVNT', 'ACTVTY', and 'ACTN') to avoid this problem.

3. <u>Values</u>

Attributes may take on many different kinds of values. These values may be considered "quantitative", "qualitative", "absolute", "relative", etc. For example, the weight of a man may be given as "155 pounds", "heavy", "heavier than John", "more than 155 pounds", or "somewhere from 150 pounds to 160

pounds", just to mention a few. For the description of queuing problems it was found to be convenient to formally categorize the various kinds of attribute values. Figure 2.5 shows graphically the superset structure resulting from this categorization, using the same notational conventions that were used in Figure 2.4.

It can be seen that all named records there are in the set 'VALU'. Seven basically different kinds of values are considered: quantitative values ('QUANVAL'), qualitative values ('QUALVAL'), relative values ('RELVAL'), standard distributions ('STDIST'), relational type values ('RELTV'), functions ('FNCTN'), and "argument" values ('ARGVAL'). Each of these is then broken down further.

Perhaps the most important kind of value in queuing problems is the QUANVAL, and within it the absolute quantity ('ABSQNTY') and the absolute time ('ABSTIME'). The set ABSQNTY consists mostly of UNIT's and DECIMAL's. The dotted lines in the figure indicate that there are other named records with the same SUP that are not shown. For example, the records 'TWO', 'THREE', 'FOUR', etc. have a SUP of 'UNIT' also, as may be seen in Appendix B. The numbers appearing in parentheses are values of the NUM attribute of these named records and give the numerical value associated with the name. For the DECIMAL's NUM is in parts per thousand. The records in the sets ABSTIME and ABSWEIT define the names of some units of measurement of time and weight, and also give some conversion information, as can be seen in the appendix.

In this figure, in addition to showing the value of the PS attribute of a record in lower case letters, the value of the ATTRIB attribute is also shown this way. This should not cause any confusion, however, because PS is always the name of a part-of-speech, whereas ATTRIB is the name of an attribute (e.g. weight, color, size). ATTRIB is used by the English decoding rules to determine which attribute is being specified in a sentence like "The car is blue." or "The car is fast." It could also be used to disallow a sentence like "The weight of the car is three minutes."

The records in the sets QUALVAL and RELVAL which appear next should be fairly self-explanatory at this point. Then, there are three records in the set STDIST -- 'UNIFORM', 'EXPON', and 'NORMAL'. These correspond to the uniform, exponential, and normal probability distributions. As will be discussed in a later chapter, a rule in the English decoding morphology is required to recognize "exponential", because it consists of more than eight characters, and the names of named records are limited in length to eight characters. The LY indicator in these records tells the decoding and encoding rules that "ly" can be added to the adjective form to make an adverb related to the same meaning. Distribution type values are very important in queuing problems. For example, the DURATION attribute of a 'SERVIC' action may have the value "exponential, mean = ten minutes", which could be specified with a sentence like "Service times are exponentially distributed with a mean of ten minutes." (Actually, distribution

values may themselves be considered as objects to be described by their attributes, such as MEAN, RANGE, and STDEV. These attributes would also take on values from some of the subsets of VALU. This whole idea will be fully discussed in Section C of this chapter.)

There are two groups of relational type values ('RELTV'). The first ('RELIND1') consists of quantitative relations, such as "less than" or "equal to", and the second ('RELIND2') consists of qualitative relations -- the comparative and the superlative. In application, each of these requires a modifier to specify what the comparison is being made to, as will be seen in an example later.

Values in the set FNCTN are specified by a sequence of ordered pairs. For example, the value of the DURATION attribute of a 'SERVIC' action may be "six minutes for cars, and ten minutes for trucks," which could also be expressed as "(cars - 6 min; trucks - 10 min)" to emphasize the sequence of ordered pairs. The value in this example would be considered to be a TYPTABL; the others in this set are similar. The last type of value shown in the figure is ARGVAL. The use of the records in this set will not be discussed here, but will be seen in later examples.

4. Other Named Records

So far in this section the named records for entities, actions, and values have been discussed. These comprise about two thirds of the group of named records with information about

words and concepts for simple queuing problems, and are the only ones that exist in multi-level superset structures. The rest of the records in the group can be classified under 15 different headings, according to their SUP attributes if they have them or their PS attributes otherwise. Figure 2.6 shows this classification, with the name of each record simply listed under the appropriate heading. The actual named record definitions may be seen in the appendix.

The first group in the figure consists of the names of several attributes which are important in specifying queuing problems. The records for these names all have a SUP attribute pointing to 'ATTR', and as can be seen in the figure, the 'ATTR' record has a PS attribute of 'NOUNS'. This means these records are all in the set ATTR, and their names can be treated as noun stems when they appear in English text. This, along with the scheme for the implicit naming of attributes discussed earlier, makes it possible to process sentences like "The color of the car is blue" and "The man's weight is 155 pounds" in a very straightforward fashion, as will be seen in the next two chapters of this report.

The next group of records shown in the figure are called attribute verbs because they appear in clauses which have as their purpose the specification of attribute values, as opposed to stating an action. For example, the sentence "The dock can hold two ships." would specify the value of the CAPACITY attribute of some dock. In the previous paragraph a similar use

of "is" (which is just a form of 'BE') was seen. The records 'IS', 'ARE', 'HAS', and 'HELD' in this group are for irregular verb forms, similar to 'LEFT' and 'TOOK' which were discussed earlier.

The next ten groups really do not require any detailed explanation. Most of those records are included merely to give part-of-speech information that is needed during decoding. The final three groups will not be explained here, either, but will be discussed in detail in the next section of this chapter.

The information about words and concepts which has been described in this section is intended to be illustrative rather than exhaustive. Clearly, much more information of this sort would be required if a larger class of queuing problems were to be handled. Some of this would simply be vocabulary expansion, which would require minimal effort, whereas some of it might involve establishing additional concept structures. As will be seen later, any significant expansion would necessitate writing additional encoding and decoding rules, also. Of course, for a completely different application of the system, an entirely different set of named records with information about words and concepts would probably be required. Experience to date has indicated that the manner of specifying this information in this system is very flexible and powerful and quite natural to use

C. THE INTERNAL PROBLEM DESCRIPTION

The Internal Problem Description (IPD) is the structure used to hold the information about any particular queuing problem. Whereas information about words and concepts is entered into the system by named record definitions, an IPD is built through the process of decoding an English problem description, as will be discussed in detail in Chapter IV. It consists of records in the cell structure also, but these records do not have NAME attributes (except for the list records, which will be described shortly). Whereas the named records described in the previous section may be considered to hold facts about queuing problems in general, an IPD holds facts about a specific queuing problem. This distinction may be roughly equated to that between the information given in the body of a textbook and that given in a homework problem at the end of a chapter. As might be expected, however, an IPD has many connections into the concept structure already described.

This section begins with a discussion of the overall structure of the Internal Problem Description and then describes each of the various kinds of records which may be found in an IPD produced by the current set of decoding rules. The IPD for the initial sample problem given in the Introduction is used as an illustration throughout.

1. Overall Structure

There is a record in the IPD for each entity in the problem. Some of these are physical entities, but most of them are what might be called abstract entities. The physical entities include

mobile entities, stationary entities, and miscellaneous physical entities such as cargo. The abstract entities include actions, distributions, successor descriptors, location descriptors, relation descriptors, quantitative values, and conditions.

Each of these entity records in the IPD is a member of one of seven lists. There is the action list ('ACTNLIST'), the mobile entity list ('MOBLIST'), the stationary entity list ('STALIST'), the distribution list ('DSTRLIST'), the successor descriptor list ('SCSRLIST'), the miscellaneous list ('MISCLIST'), and the unit list ('UNITLIST'). Each of these lists is a named record with a SUP of 'RECLIST'. The items on a list appear as the values of attributes 11, 12, 13, etc. (i.e. these attributes are type-3 cells pointing to the entity records.) Each list record also has a LASTREC attribute whose value is the number of the highest pointer attribute being used in that record. For example, if there were 5 actions in a problem, LASTREC('ACTNLIST') -- read "LASTREC of action list" -- would have the value 15.

The named record definitions for each of these lists can be seen at the end of Appendix B. These definitions serve the purpose of initializing the lists, with a LASTREC of 10 indicating that a list is empty. As entities are encountered during the decoding of an English problem description, records are created to represent them and pointers to these new records are added to the appropriate lists, with the LASTREC attributes being incremented accordingly. (More will be said about this in Chapter IV.

It can also be seen at the end of Appendix B that there are attributes of the MEMORY record set to point to each of the list

records. This is not really necessary, but it does serve to "tie everything together," making it possible to get to every record in the IPD through MEMORY. The MEMORY record also has two other attributes, PROBTIME (problem time) and TIMUNIT (time unit), which get their values during decoding. These point to quantitative value records in the set 'ABSTIME' (i.e. records whose SUP chains lead to the named record 'ABSTIME') and specify the length of time for which the simulation is to be run and the basic time unit to be used in the GPSS program.

Figure 2.7 (two pages) lists the attributes that each type of IPD record has, with the kinds of values that each attribute can have being listed in parentheses after the attribute name. A value of "number" would be a type-0 cell, a "name" would be a type-1 cell, and all others would be type-3 cells, i.e. pointers. A dollar sign ($) means "in the set" (i.e. "with SUP chain leading to"), as discussed previously in Section B.1. Currently, none of the IPD records have indicators, except temporarily during decoding and encoding. MEMORY and the list records shown in the figure have already been described, and the other records shown there will be described in the following subsections.

From the description of these records, it will be apparent that the form of the IPD has been greatly influenced by the fact that it is to be used to produce a GPSS program. Because the IPD serves as sort of a "middle ground" between the English input text and the GPSS program, its design involved considering the tradeoffs in difficulty between doing certain processing as part of English decoding and doing it

MEMORY Record

ACPTR('ACTNLIST'), MEPTR('MOBLIST'), SEPTR('STALIST'), DISTPTR('DSTRLIST'),SUCPTR('SCSRLIST'),MISCPTR('MISCLIST.), UNITPTR('UNITLIST'), PROBTIME($'ABSTIME'), TIMUNIT($'ABSTIME')

List Records

SUP('RECLIST'), NAME("ACTNLIST","MOBLIST",etc.), LASTREC(number), @11, @12, etc. (pointers to entity records)

Action Records

SUP($'ACTION'), IDNO(number), AGENT($'MOBENTY'), GOAL($'MOBENTY',$'CARGO'), MOBENATR('AGENT','GOAL'), LOCATION($'LOCDESCR'), IETM($'ABSTIME',$'STDIST',$'TYPTABL'), DURATION($'ABSTIME',$'STDIST',$'TYPTABL',$'COND'), SUCC($'ACTION',$'SUCDESCR'), ASNDISTR($'TYPDIST')

Mobile Entity Records

SUP($'MOBENTY'), IDNO(number), IDNAME(name),CONSUMP($'UNIT'), STRUC($'MOBENTY'), CLASATR($'ATTR','SOUP'), SIZE($'RELSIZ'), COLOR($'ABSCOLR',$'RELCOLR'), WEIGHT($'ABSWEIT',$'RELWEIT'), SPEED($'RELSPEED'), TYPE($'ABSTYPE')

Stationary Entity Records

SUP($'STATENTY'), IDNO(number), IDNAME(name), QUANTITY(number), CAPACITY($'UNIT'), STRUC($'STATENTY'), CLASATR($'ATTR','SOUP'), LOCATION($'LOCDESCR'), SIZE($'RELSIZ'), COLOR($'ABSCOLR', $'RELCOLR'), WEIGHT($'ABSWEIT',$'RELWEIT'), SPEED($'RELSPEED'), TYPE($'ABSTYPE')

Distribution Records

Standard Distribution

SUP($'STDIST'), IDNO(number), MEAN($'ABSTIME',$'TYPTABL', $'NORMAL'), RANGE($'ABSTIME',$'NORMAL'), STDEV($'ABSTIME', $'TYPTABL',$'NORMAL')

Type Table

SUP('TYPTABL'), IDNO(number), FNARG('PARAM1'), DORC("D","C"), XYLAST(number), @101, @103, etc. ($'MOBENTY'), @102, @104, etc. ($'QUANVAL')

Type Distribution

SUP('TYPDIST'), IDNO(number), FNARG('RANDMREC'), DORC("D"), PNUM('ONE'), XYLAST(number), @101, @103, etc. ($'DECIMAL'), @102, @104, etc. ($'MOBENTY')

Figure 2.7. Kinds of Records in the Internal Problem Description (cont'd on next page)

<u>Successor Descriptor Records</u>

<u>Fractional</u>

SUP('FRACTNL'), IDNO(number), FNARG('RANDMREC'), DORC("D"), XYLAST(number), @101, @103, etc. ($'DECIMAL'), @102, @104, etc. ($'ACTION')

<u>Parameter Type</u>

SUP('PTYP'), IDNO(number), FNARG('PARAM1'), DORC("D"), XYLAST(number), @101, @103, etc. ($'MOBENTY'), @102, @104, etc. ($'ACTION')

<u>Queue, Facility, and Storage Types</u>

SUP('QTYP','FTYP','STYP'), SUCARG('STATENTY'), OPENACT($'ACTION',$'FRACTNL',$'PTYP'), CLOSACT($'ACTION',$'FRACTNL',$'PTYP'), MAXQ($'UNIT')

<u>Miscellaneous Records</u>

<u>Location Descriptor</u>

SUP($'LOCDESCR'), LOCOBJ($'STATENTY')

<u>Relation Descriptor</u>

SUP($'RELIND1'), OBJREL($'QUANVAL',$'ENTITY')

<u>Quantitative Value</u>

SUP($'QUANVAL'), NUM(number)

<u>Condition</u>

SUP($'COND'), CONDENTY($'STATENTY')

<u>Miscellaneous Entity</u>

SUP($'ENTITY'), LOCATION($'STATENTY'), LENGTH($'RELIND1')

<u>Unit Records</u>

SUP('UNIT'), NUM(number), NAME("ONE","TWO",...,"TEN")

Figure 2.7. Kinds of Records in the Internal Problem Description (cont'd from previous page)

as part of GPSS encoding. The form of the IPD tends to lean a bit more towards GPSS because the GPSS encoding rules were written before the English decoding rules.

A graphic portrayal of the IPD for the sample problem given in Figure 1.1 in the Introduction appears in Figure 2.8. In this figure each record of the IPD is represented by a box, with a name appearing at the top of each box. With the exception of 'ACTNLIST', which is the only named record shown in the figure, these names would not actually exist within the computer, but were placed on the drawing simply to furnish a means of referring to the various records in the discussion which follows. In each box the attribute-value pairs of the record are shown, with the attribute name or number on the left and its value on the right. Many of the values are pointers to other records in the IPD, in which case an appropriate arrow is drawn.

The only list record included in the drawing is 'ACTNLIST', with the others being left off in order to keep the number of lines at a minimum. However, it is important to note that every record in the figure (except MEMORY,'ACTNLIST', and REC11-13) is a member of some one of the other six lists and should actually have an additional pointer to it from the appropriate list record. It can be seen in the figure that every entity record, except the initial "arrive", serves as the value of (i.e. is pointed to by) some attribute of another entity record. For example,a mobile entity is pointed to from either the AGENT or GOAL attribute of each action it is involved in. Each record in the

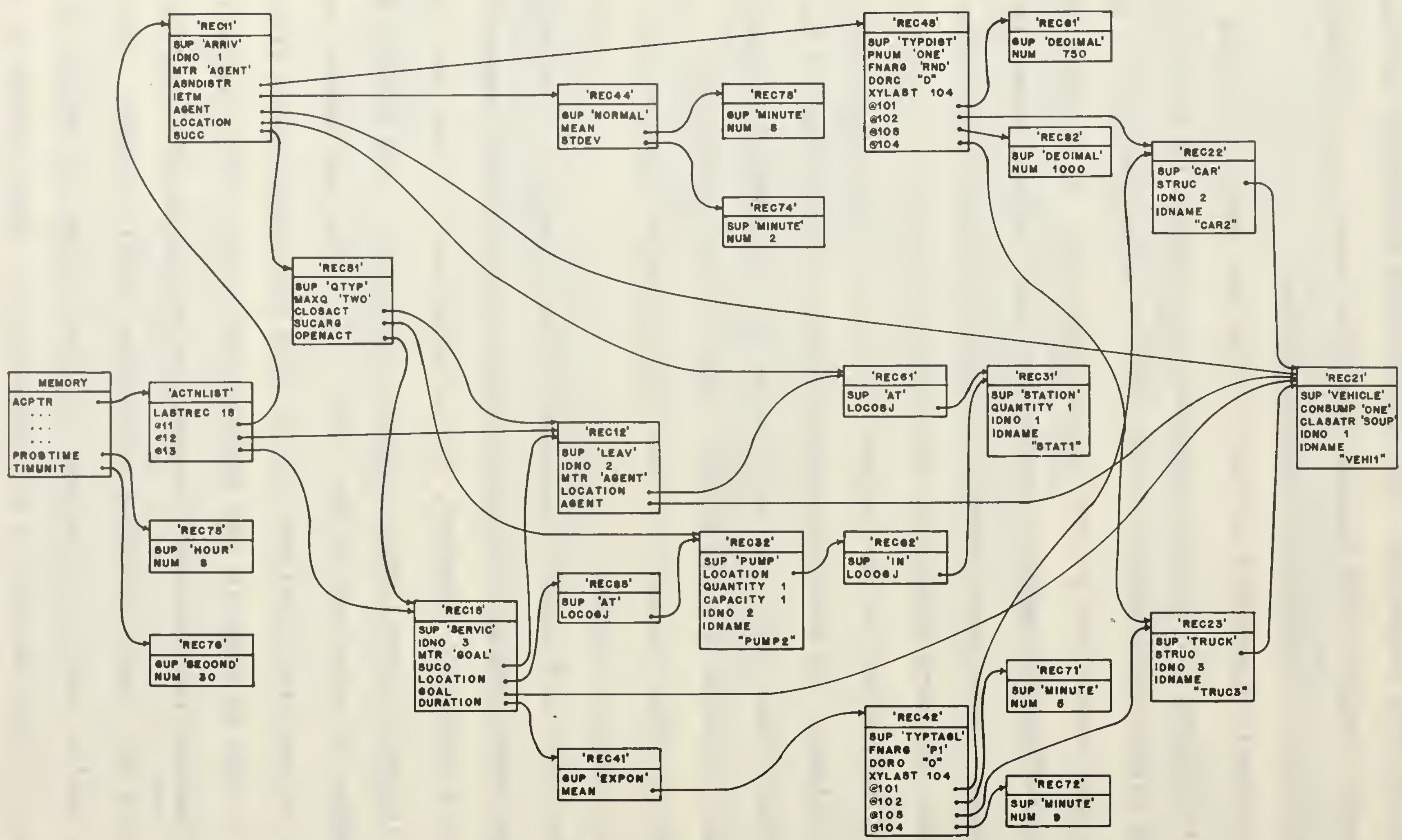


Figure 2.8. The Internal Problem Description for the Sample Problem of Figure 1.1

drawing will be referred to for illustrative purposes at the appropriate point in the discussion in the following subsections.

Every IPD record has a SUP attribute pointing to the named record representing the concept of which this record is a specific instance. For example, the SUP of the first action record (REC11) in Figure 2.8 points to the named record 'ARRIV', indicating that this action (vehicles arrive at a station) is a specific instance of the concept "arrive." This use of the SUP is slightly different from that discussed earlier, but yet basically the same. As will be seen later, it is very important and useful during both decoding and encoding.

It can be seen in the figure that some other attributes of IPD records have values which are pointers to named records, too. For example, the MTR attribute of REC11 points to the named record 'AGENT'. Perhaps a better example, but one which does not appear in the figure, would be the COLOR attribute of a mobile entity record pointing to the named record 'BLUE'.

Figure 2.9 is an attempt to show graphically the relationship between the concept structure, represented by the named records, and the IPD, for one small portion of the IPD of Figure 2.8. Unfortunately, it is not practical to do this on any larger scale than is done there because to be complete most of the 300 named records discussed in Section B of this chapter would have to be included in the concept structure. It is convenient to think of the concept structure as existing in one or more planes perpendicular to the plane of the IPD. For

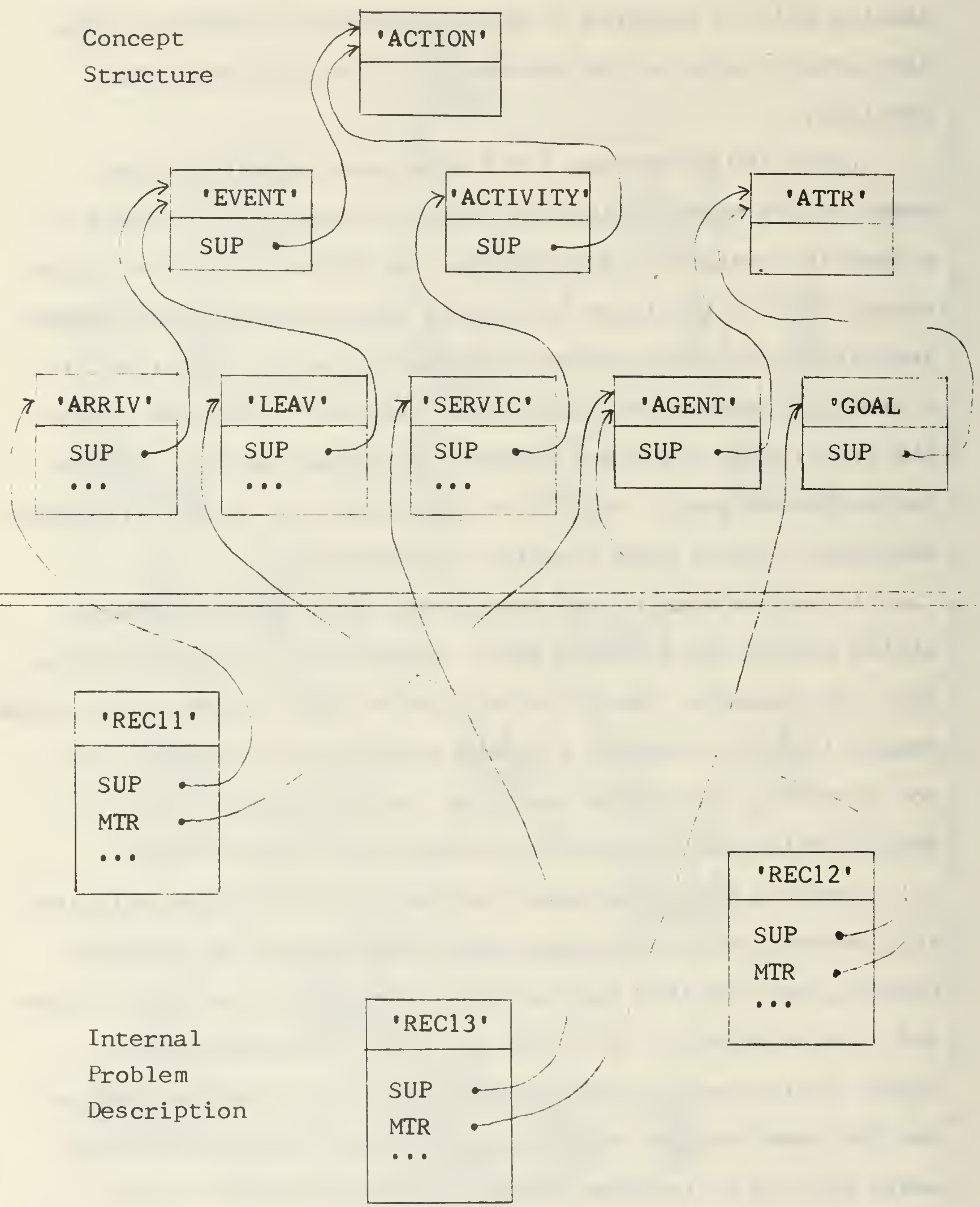


Figure 2.9. Relationship between the Concept Structure and the Internal Problem Description

understanding of the remainder of this report it is essential to keep in mind a picture of the relationship between the concept structure and the IPD.

There is actually another connection between IPD records and the concept structure which should be mentioned here, although its importance and usefulness probably will not be recognized until later when encoding and decoding are discussed. This has to do with implicit naming of attributes (i.e. implicit association of a name with an attribute number) which was discussed in Section A.6. An attribute number greater than 300 is actually the identification number of a named record (e.g. 'AGENT'), and therefore may itself be considered to be pointing into the concept structure in some cases.

2. Action Records

The action records are probably the most important entity records in the IPD, because the problem description is really built around the actions that take place in the queuing system, and all the other entity records may be considered to be simply furnishing information needed to fully specify the actions. The fact that the action records in Figure 2.8 (REC11-13) appear pretty much to the left in the figure is indicative of this. The attribute requirements of action records are summarized along with the others in Figure 2.7, and three examples of these records appear in Figure 2.8, as just mentioned. Action records are described in this subsection, and the various kinds of records they point to are described in the following subsections.

In addition to the SUP attribute, which has already been discussed in general for all IPD entities, each action record must have either an AGENT or a GOAL which points to a mobile entity record (i.e. an IPD record which has its SUP pointing to a named record in the set 'MOBENTY'). The AGENT of an action is the one which does the action, and the GOAL is the one to which the action is done. The MOBENATR attribute tells which of these two attributes is pointing to a mobile entity. The GOAL may point to some miscellaneous physical entity (e.g. with a SUP of 'CARGO'). In Figures 2.8 and 2.9 MOBENATR is abbreviated as MTR.

Each action record must also have a LOCATION pointing to a location descriptor record, which in turn points to a stationary entity record. An event with a SUP of 'ARRIV' or 'ENTER' must have an IETM (inter-event time) attribute to specify the time between occurrences of the event, and an activity (e.g. with a SUP of 'SERVIC' or 'LOAD') must have a DURATION attribute to specify the time taken to perform the activity. These times can be given as constants, standard probability distributions, functions, or combinations of these, some of which can be seen in Figure 2.8. Also, the DURATION can be specified conditionally, to depend upon the status of some piece of equipment at simulation time.

Each action record, except a 'LEAV' (i.e. one with a SUP of 'LEAV'), must have a SUCCessor attribute to specify which action the mobile entity of this action is involved in next. The value of SUCC may simply be a pointer to another action record, as in

REC13 of Figure 2.8, or it may be a pointer to a successor descriptor record, as in REC11. An 'ARRIV' or an 'ENTER' may have an ASNDISTR (assignment distribution) attribute to specify the percentages of the various kinds of mobile entities which arrive or enter, in the form of a cumulative probability distribution. This is tied in with the CLASATR and STRUC attributes of the mobile entities, as will be discussed in the next subsection. Each action record also has an IDNO (identification number) attribute which is assigned when the action is first encountered during decoding. IDNO is needed when encoding the GPSS program.

3. Mobile Entity Records

The only attributes which a mobile entity record must have are SUP, IDNO, and IDNAME, plus it must have access to a CONSUMP attribute, either by having one itself or by there being one in a record in its STRUC chain. (STRUC will be discussed shortly.) IDNO is assigned and used similarly as it is for action records. The value of IDNAME is formed for symbolic naming purposes as part of the GPSS encoding process by concatenating the first three or four letters of the NAME of the SUP of a record with the value of its IDNO. CONSUMPtion indicates how many units of a resource are required by the mobile entity, and is used for the B argument in ENTER and LEAVE blocks involving this mobile entity in the GPSS program. Currently, there is no provision for specifying that a particular mobile entity in a problem requires different amounts of different resources.

The attributes SIZE, COLOR, WEIGHT, SPEED, and TYPE are included in Figure 2.7 just to show some of the possibly important attributes that mobile entities in general might have. Currently, these are just being used to hold distinguishing features for the English problem description (e.g. to make it possible to refer distinctly to "big ships" or "small ships"). Actually, as will be discussed in Chapter IV, the decoding end of this has not been completely implemented yet.

The CLASATR (class attribute) and STRUCture attributes go along with this idea of distinguishing features. For example, if a problem description included "red cars", "white cars", and "blue cars", there would be one IPD record to represent all of the cars, plus three other records to represent each kind (i.e. color) of car. All four records would have a SUP of 'CAR', and each of the last three records mentioned would have a COLOR attribute pointing to the named record for the particular color and a STRUC attribute pointing to the IPD record representing all of the cars in the problem. The record representing all of the cars would have a CLASATR attribute with the value 'COLOR' (i.e. a pointer to the named record 'COLOR'), to indicate what is the distinguishing attribute of those records which have a STRUC attribute pointing to this record. Conceptually, these structures could be multi-leveled (therefore the term "STRUC chain"), but some aspects of the current implementation are limited to one level.

A special case of the above can be seen for the mobile entity records of Figure 2.8 (REC21-23). There the distinguishing

attribute is SUP. (i.e. Cars and trucks are particular kinds of vehicles, but they have their own distinct names, rather than being referred to, say, as "small vehicles" and "big vehicles.") Because, as discussed earlier, SUP has been declared explicitly as the name of attribute number 1, there could be some ambiguity by having a named record with the same name. Therefore, a CLASATR of 'SOUP' is used to indicate this special case. Part of the usefulness of the STRUC attribute is that it avoids some unnecessary duplication of information. For example, the value of CONSUMP is the same for both cars and trucks in the sample problem, so it need be stored only once, "up" in the record for vehicles. Currently, CLASATR and STRUC attributes are established for a mobile entity when a type distribution is built as the value of an ASNDISTR attribute of some action.

4. <u>Stationary Entity Records</u>

As might be expected, stationary entity records are quite similar to those just described for mobile entities. The only attributes which a stationary entity record must have are SUP, IDNO, and IDNAME, plus it must have access to QUANTITY and possibly CAPACITY attributes, either by having them itself or by having them in some record in its STRUC chain. IDNO and IDNAME are treated the same as for mobile entities.

The QUANTITY attribute specifies how many entities of this type exist in the queuing system, and CAPACITY specifies how many "resource units" each one has. As pointed out in the previous subsection, each mobile entity record has a CONSUMP attribute to specify how many resource units are required by

entities of that type. QUANTITY and CAPACITY are used together to determine the capacity of a STORAGE in the GPSS program. A missing CAPACITY attribute indicates unlimited capacity.

As can be seen in Figure 2.7, a stationary entity record may have a LOCATION attribute, which would point to a location descriptor record just as the LOCATION of an action does. It can also be seen that it may have the same optional attributes already discussed for mobile entities, and that discussion is applicable here. In Figure 2.8 REC31 and REC32 are stationary entity records.

5. Distribution Records

There are basically three different kinds of records which may be found on the distribution list -- standard distributions, type tables, and type distributions -- as can be seen in Figure 2.7. A standard distribution record has a SUP in the set 'STDIST'. Currently, this set consists only of 'UNIFORM', 'EXPON', and 'NORMAL', as discussed in section B.3, but it could easily be expanded. The values of the parameters of any particular standard distribution (e.g. an exponential with a mean of 8 minutes) are given by the attributes of the record representing it, with these attributes always being pointers to other IPD records.

All three standard distributions require a MEAN. Additionally, a 'UNIFORM' requires a RANGE (actually the half-range) and a 'NORMAL' requires a STDEV. Currently, a meaningful value for any of these attributes may be some 'ABSTIME' (e.g. 8 minutes), a 'TYPTABL' (except for RANGE), or a 'NORMAL' (i.e. the para-

meters themselves may be normally distributed, to any depth). Any apparent lack of generality in the above is due to peculiarities of the GPSS language. Each normal distribution is also given an IDNO during the GPSS encoding process to serve as the number of the FVARIABLE defining it in the program. REC41 and REC44 in Figure 2.8 are standard distribution records. Although this kind of record has been used only as the value of IETM and DURATION so far, it is obvious that it could find use in specifying the value of any attribute that can take on quantitative values (e.g. WEIGHT).

A type table can be used to specify the value of an attribute when the value depends on the mobile entity involved. For example, in the sample problem the mean of the exponential distribution of service times is different for cars and trucks. As can be seen in Figure 2.8, this mean is specified by a 'TYPTABL' (REC42) which is really a function with the records for cars and trucks as its X values and the records for 5 minutes and 9 minutes as its Y values. In the GPSS program a 'TYPTABL' is realized as a FUNCTION, as are certain other records to be discussed shortly.

All of the records that are realized as GPSS FUNCTION's have essentially the same attributes. The X values are pointed to from the odd-numbered attributes starting at @101, and the Y values are pointed to from the even-numbered ones starting at @102, with XYLAST giving the number of the last attribute used. DORC (D or C) has the value "D" if the FUNCTION is to be discrete, and "C" if it is to be continuous, using linear interpolation.

FNARG specifies the argument of the FUNCTION, and currently would always be 'PARAM1' (abbrev. 'P1') for a 'TYPTABL', because in the GPSS program parameter 1 of a transaction is used to hold the identifying number (from IDNO) of the type of mobile entity which the transaction represents. Function records are given IDNO's during GPSS encoding, also. As can be seen in Figure 2.7, several different attributes can take 'TYPTABL' values, and the principle is probably even more generally applicable.

A type distribution specifies, in the form of a cumulative probability distribution, the percentages of the various kinds of mobile entites within a class. For example a 'TYPDIST' might give the percentages of red cars, white cars, and blue cars. REC43 in Figure 2.8, for another example, specifies that 75 percent of the vehicles are cars and 25 percent are trucks. As can be seen, the Y values point to the mobile entity records. Currently, only an ASNDISTR (of an action) can have a 'TYPDIST' for its value.

A type distribution is realized as a GPSS FUNCTION and, therefore, has the attributes described above. The FNARG is 'RANDMREC' (abbrev. 'RND') to indicate that the argument of the FUNCTION is a random number (e.g. RN2). The PNUM attribute specifies which transaction parameter is to be used to hold the identifying number of the type of mobile entity which the transaction represents. Parameter 1 is currently the only one used for this.

6. Successor Descriptor Records

As was mentioned earlier, the SUCC attribute of an action record may point to a successor descriptor record. This is required when the action is not always followed by one particular action, but rather may be followed by any one of several actions, depending upon something. The successor descriptor record lists the alternative actions and contains information about how the choice is to be made.

Five types of successor descriptors have been implemented. Two of these ('FRACTNL' and 'PTYP') allow any number of alternative actions and are realized as GPSS FUNCTION's. A 'FRACTNL' is intended for the situation where the successor action is simply to be chosen randomly, and what it consists of essentially is a cumulative probability distribution with pointers to the alternative actions as the Y values. An example of a 'FRACTNL', stated in English, is "Then, 40 percent of the vehicles are washed, 35 percent get air, and 25 percent leave."

A 'PTYP' is intended for the situation where the successor action depends on the particular type of mobile entity. It is a function with pointers to the mobile entity records as the X values and pointers to the actions as the Y values. An example of a 'PTYP', stated in English, is "Then, red cars are washed, white cars get air, and blue cars leave," or "Then, cars are washed and trucks leave." 'FRACTNL' and 'PTYP' records have the usual function attributes described in the previous subsection.

The other three types of successor descriptors which have been implemented ('QTYP', 'FTYP', and 'STYP') allow only two alternative actions, with the choice between the two depending

on the presence or absence of some condition in the simulated system. REC51 in Figure 2.8 is a 'QTYP', and can be interpreted as saying, "Then, if the length of the line at the pump (SUCARG) is less than two (MAXQ), be serviced (OPENACT); otherwise, leave (CLOSACT)." An example of an 'FTYP', stated in English, is "Then, if the pump (SUCARG) is not busy, be serviced (OPENACT); otherwise, leave (CLOSACT)." An example of an 'STYP', stated in English, is "Then, if the station (SUCARG) is not full, be serviced (OPENACT); otherwise, leave (CLOSACT)."

The SUCARG in the 'QTYP', 'FTYP', and 'STYP' records must point to a stationary entity record. The 'FTYP' is used with a stationary entity that has a capacity of one (a facility in the GPSS program), and the 'STYP' is used with a stationary entity that has a capacity of more than one (a storage in the GPSS program). OPENACT and CLOSACT may point either to action records or to 'FRACTNL' or 'PTYP' successor descriptor records. The latter would be used in a situation like "When a vehicle arrives, if a pump is available, the vehicle will be serviced; otherwise, a car will just get air and a truck will leave immediately."

7. Miscellaneous Records

All of the lists discussed so far contain some records which directly result in complete GPSS statements -- blocks, table and storage definitions, variables and functions. None of the items on the miscellaneous list do this, but they are needed to provide supporting information. The main reason for keeping them on a list is to avoid unnecessary duplication during decoding.

One type of record on this list is the location descriptor, three examples of which may be seen in Figure 2.8 (REC61-63). Location descriptors serve as the value of LOCATION's, and have just two attributes, as shown in Figure 2.7 -- a SUP pointing to a named record in the set 'LOCDESCR' (Figure 2.6), and a LOCOBJ (location object) pointing to a stationary entity record. The interpretation should be obvious.

A relation descriptor is intended to hold certain relative values, such as "less than two." There is no example of this in Figure 2.8 but a record of this type would have been created during the decoding of sentence 3 of the sample problem. The value of its OBJREL became the value of MAXQ in REC51.

Quantitative values are probably the most numerous of the various kinds of records on the miscellaneous list, and probably most of these are in the set 'ABSTIME' because of the important role which time plays in queuing system simulations. Each of these records has a SUP pointing to the named record representing the appropriate unit and a NUM to tell how many, as can be seen in Figure 2.8 (REC71-76). Some quantitative value records represent fractions, and have a SUP of 'DECIMAL' and a NUM which is considered to be in parts per thousand. REC81 and REC82 in Figure 2.8 furnish examples of this. Quantitative value records would also be used to represent quantities such as "10 tons" or "5 feet."

A condition record serves as the value of the DURATION attribute of an action in the case that the duration depends on the status of some piece of equipment in the system. For example,

"a car waits until the pump is available" would require a condition record with a SUP of 'COND1' and a CONDENTY pointing to the stationary entity record representing the pump. This condition record would be pointed to from the DURATION attribute of the record representing the action "cars wait." A SUP of 'COND2' is used when the condition is the opposite of the one given in the above example, i.e. "not available."

A miscellaneous entity record may be used during decoding to represent the "line" in a sentence like number 3 in the sample problem. In this case it would have LENGTH and LOCATION attributes. If "cargo" were mentioned in a problem, it would result in a miscellaneous entity record also.

8. Unit Records

Unit records are used to represent dimensionless integer quantities. Records of this type are very important during decoding and encoding, but they usually do not appear in the IPD itself. Each has a SUP of 'UNIT' and a NUM with an integer value. Records for the numbers 1 through 10 are entered as named records, and 'UNITLIST' is initialized to point to them, as can be seen at the end of Appendix B. This makes it possible to refer to these numbers by name in English text. When a new number (e.g. "27") is encountered during decoding of an English problem description, an appropriate unit record is created (without a NAME) and added to the list.

D. SUMMARY AND DISCUSSION

The first section of this chapter described the manner in which information is held in the entity-attribute-value form in this system <u>for any application</u>. This included a discussion of how to enter information by means of named record definitions. Named records are intended primarily to hold static information about both the words and the concepts relevant to some particular application of the system. This information is used by the decoding and encoding rules during the processing of text, as will be described in detail in the next two chapters, with its exact form depending on the way it is to be used by a particular set of rules. (To a large extent the writing of named record definitions and the writing of rules go "hand-in-hand.")

The second section described a particular group of named records currently being used to furnish the information about words and concepts for simple queuing problems. The discussion was mostly concerned with those records in the sets 'ENTITY', 'ACTION', and 'VALU', which are highly structured.

In the last section the Internal Problem Description was discussed. This is the record structure built during decoding to hold the information about a particular queuing problem. The overall form of the IPD was discussed first, and then each of the various kinds of records which may be found in an IPD produced by the current set of decoding rules was described.

Just as the group of named records discussed in Section B was intended to be illustrative rather than exhaustive, so were

the IPD records discussed in Section C. These kinds of records would be adequate for describing many queuing problems, but certainly there are many, many more problems for which they would be inadequate. However, it appears that the overall structure of the IPD and the groupings of records within it are appropriate, and that a good deal of expansion for handling a larger class of queuing problems could be accomplished in a fairly straightforward fashion.

The relationship between the IPD and the concept structure, as discussed in Section C.1, is considered to be especially significant. This relationship comes about rather naturally because of the fact that in this system general information is stored in exactly the same manner as specific information (i.e. as records in the cell structure). Benefits are derived from this during both encoding and decoding, as will be seen in the next two chapters.

III. ENCODING

In the system being described here, encoding is the process of translating information which is in the form of records in the cell structure into equivalent information in the form of text. The exact manner in which encoding is to be done for any particular application is specified by sets of encoding rules written in the rule language of the system. After these rules have been compiled into their internal computer format by a FORTRAN routine, they are used by the encoding algorithm, which is another FORTRAN routine, to produce output.

This chapter begins with a description of the encoding process, including the rule language, without regard to any particular application of the system. Then there is a section describing the sets of encoding rules which have been written for the queuing problem application, especially those for producing English text descriptions and those for producing GPSS programs.

A. THE ENCODING PROCESS

The approach which appears to be the most widely used for having a computer produce sentences and phrases in a natural language such as English is the "canned message." This technique is usually used by compilers and other programs to give directions to the user and to inform him of errors. In its simplest form it consists of having a number of completely prepared messages stored in memory to be printed when

appropriate. Sometimes a message may have a "slot" within it into which some variable piece of information may be inserted just prior to printing. The canned message approach is quite practical and reasonable to use in a situation where the set of messages to be produced is small.

It seems fairly obvious that human beings do not communicate solely by means of canned messages. Even though each of us may have some "stock phrases" which we use quite often, nevertheless we are capable of making (and understanding) utterances that we have never made (or heard) before. We seem to have some system of rules which we follow, although usually not consciously, to generate (and understand) meaningful utterances. Attempts at discovering these rules, writing them down, and teaching them to others are made under the subject heading "grammar." Nobody has yet been able to discover <u>all</u> of the rules for any natural language, nor has anybody yet devised an adequate method for writing them down.

One type of rule which is a basic part of many of the methods proposed for describing languages is the "phrase structure rule." Because the encoding rules of this system are basically phrase structure rules, the first part of this section is devoted to a discussion of rules of this type and how they can be used to generate text. This is then followed by a detailed discussion of encoding rules and how they are used to produce text in this system.

1. <u>Phrase Structure Rules</u>

The following is a typical phrase structure rule:

SENT ::= NOUNPH VERBPH .

This rule says that a sentence consists of a noun phrase, followed by a verb phrase, followed by a period. Those symbols, such as the period, which appear in a rule and would also appear in a string in the language being described are called "terminal" symbols. The others, such as SENT, NOUNPH, and VERBPH, are called "non-terminal" symbols. The ::= means "consists of" or "is defined as".

A set of phrase structure rules may be written to describe a language or some portion of a language. Figure 3.1 shows a set of phrase structure rules for describing a very small portion of the English language. As can be seen there, each rule has a non-terminal symbol on the left and one or more symbols (both terminal and non-terminal) on the right. The same non-terminal symbol may appear on the left of more than one rule, as do VERBPH, NOUNPH, and others. There must be one non-terminal symbol, called the "goal" symbol, which does not appear on the right in any rule, SENT in this example.

It can also be seen in the figure that the same symbol may appear on both sides in a given rule, i.e. a symbol may be defined in terms of itself. This is called "recursion". There is left-recursion in rule 2 and right-recursion in rules 3 and 5. Recursion makes it possible to describe an

1	SENT	::=	NOUNPH	VERBPH	.			
2	VERBPH	::=	VERBPH	NOUNPH				
3	VERBPH	::=	VERB	VERBPH				
4	VERBPH	::=	VERB					
5	NOUNPH	::=	ADJ	NOUNPH				
6	NOUNPH	::=	NOUN					
7	VERB	::=	#	VERBP				
8	NOUN	::=	#	NOUNP				
9	ADJ	::=	#	ADJP				
10	VERBP	::=	I	S				
11	VERBP	::=	A	R	E			
12	VERBP	::=	H	A	S			
13	VERBP	::=	VERBS	I	N	G		
14	VERBP	::=	VERBS	E	N			
15	VERBP	::=	VERBS	E	D			
16	VERBP	::=	VERBS	E	S			
17	VERBP	::=	VERBS	E				
18	VERBP	::=	VERBS	S				
19	VERBP	::=	VERBS					
20	NOUNP	::=	M	A	N			
21	NOUNP	::=	M	E	N			
22	NOUNP	::=	NOUNS	S				
23	NOUNP	::=	NOUNS					
24	ADJP	::=	T	H	E			
25	ADJP	::=	B	I	G			
26	VERBS	::=	B	E				
27	VERBS	::=	H	A	V			
28	VERBS	::=	L	O	A	D		
29	VERBS	::=	U	N	L	O	A	D
30	VERBS	::=	S	E	R	V	I	C
31	NOUNS	::=	S	H	I	P		
32	NOUNS	::=	B	O	A	T		
33	NOUNS	::=	B	O	Y			

Figure 3.1. A Set of English Phrase Structure Rules

<u>infinite</u> number of strings with a <u>finite</u> number of rules. (This fact is more of theoretical interest than of practical significance, however, because these strings would become indefinitely long.)

A set of rules such as given in Figure 3.1 is called a "grammar." More specifically, this one can be called a "context-free, phrase structure grammar" [2]. This particular example is a bit different from those usually given for these grammars, in that it goes to a lower level. The typical example grammar of this sort might include just the first six rules, and then assume that there is a vocabulary available to furnish VERB's, NOUN's, and ADJectives. Of course, in that case some other device would have to be introduced to handle such things as tense and number (or, as is more often the case, these considerations would be ignored). The form of this example grammar was chosen in line with its purpose of furnishing an introduction to the material to be presented on encoding rules.

Certain naming conventions used in the rules of this example have also been used in the English encoding and decoding rules developed in this research, and should be noted at this time. PH appearing at the end of a symbol means "Phrase" (e.g. VERBPH should be read "verb phrase"), P appearing at the end of a symbol means "part" (e.g. VERBP is "verb part"), and S means "stem" (e.g. VERBS is "verb stem"). A <u>phrase</u> usually consists of more than one word (e.g. "the big ships"), a <u>part</u> is a complete word (e.g. "is" or "unloading"), and a <u>stem</u> may

take a suffix to form a complete word (e.g. "hav" or "unload"). The usual part-of-speech names (e.g. VERB) are used for a complete word plus the space ahead of it, as can be seen in rules 7 through 9. (A # in a rule denotes a space in a string.) "Word" is best defined from here on as a string of contiguous characters, including a preceding blank but not including punctuation.

The "tree" in Figure 3.2 shows how the grammar of Figure 3.1 can be used to describe the phrase structure of a simple English sentence. For example, it can be seen that the T, H, and E form an ADJP (rule 24), which when preceded by the # forms an ADJ (rule 9). The whole structure can be built up in this way to eventually form a SENT, the goal symbol. The applicable rule numbers appear in the figure, too.

1(a). Algorithms for Generating Sentences

Obviously, the grammar of Figure 3.1 could also be used to generate sentences. A simple "rewriting" algorithm to do such generation is given in Figure 3.3, and an example of its application is shown in Figure 3.4. There the initial line, SENT, is first rewritten as NOUNPH, VERBPH, and a period, using rule 1. Then NOUNPH is rewritten as ADJ and NOUNPH, using rule 5, and VERBPH is rewritten as VERBPH and NOUNPH, using rule 2. The period, being a terminal symbol, is simply copied to the next line. This procedure continues until a sentence

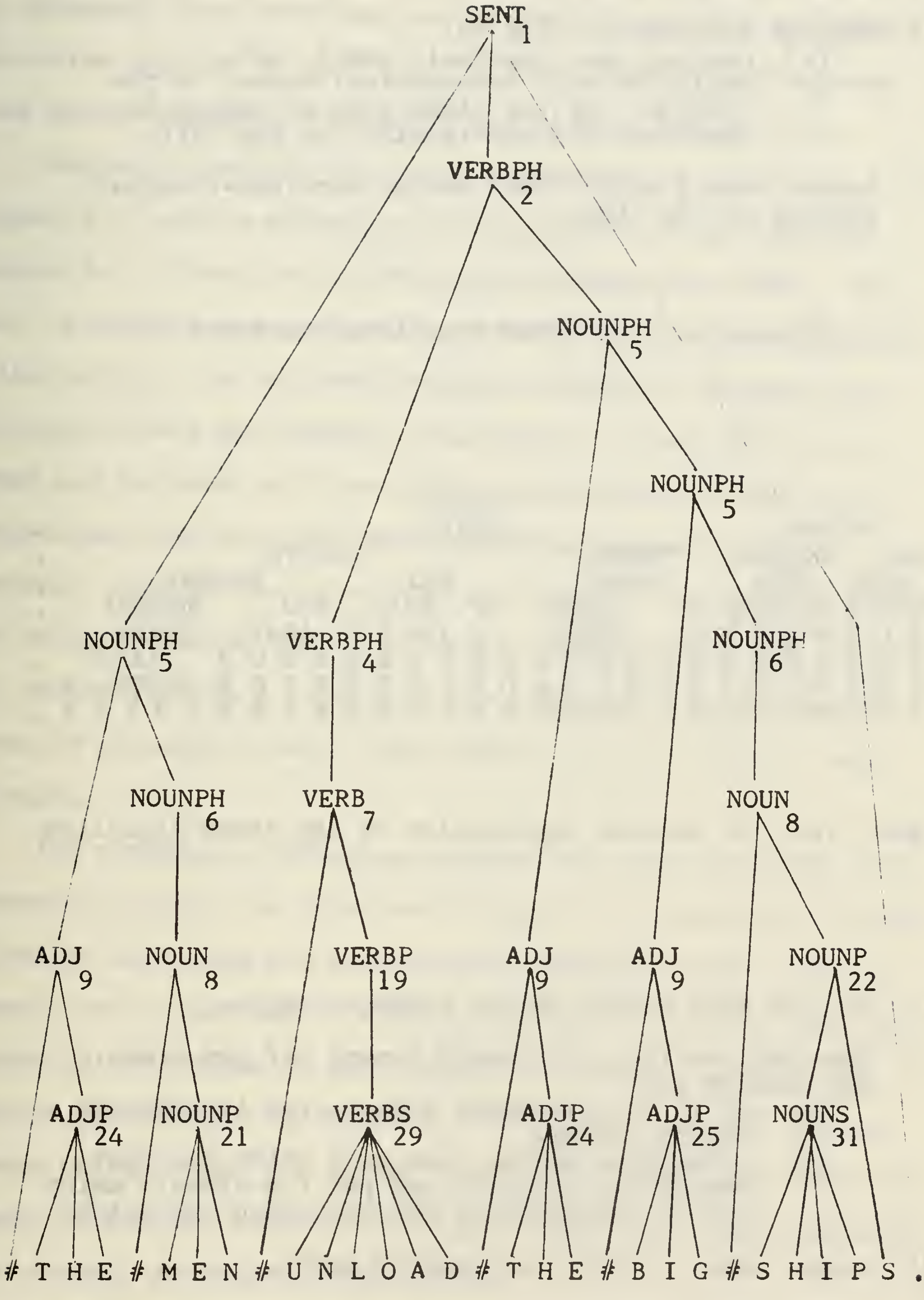


Figure 3.2. The Phrase Structure of a Simple English Sentence

(1) Write down the goal symbol to form the initial line.

(2) Rewrite the current line by:
  (a) copying each terminal symbol, and
  (b) replacing each non-terminal symbol by the symbols on the right side of some rule that has this non-terminal on the left.

(3) Repeat step 2 until there are no more non-terminal symbols in the line.

Figure 3.3. A Simple Rewriting Algorithm

```
                                  SENT
      NOUNPH                     VERBPH                          .
   ADJ    NOUNPH     VERBPH                 NOUNPH               .
#   ADJP   NOUN       VERB          ADJ           NOUNPH         .
# T H E # NOUNP #      VERBP     #  ADJP    ADJ        NOUNPH    .
# T H E # M E N #      VERBS     # T H E #  ADJP       NOUN      .
# T H E # M E N # U N L O A D # T H E # B I G #      NOUNP     .
# T H E # M E N # U N L O A D # T H E # B I G #    NOUNS   S   .
# T H E # M E N # U N L O A D # T H E # B I G # S H I P S      .
```

Figure 3.4. An Example Application of the Above Algorithm

(1) Put the goal symbol on the stack to begin.

(2) Take the top (i.e. leftmost) symbol off the stack, and examine it:
  (a) if it is a terminal symbol, put it into the output stream.
  (b) otherwise, select some rule which has that symbol on the left, and put the symbols which are on the right of the rule onto the stack.

(3) Repeat step 2 until the stack is empty.

Figure 3.5. A Rewriting Algorithm which Utilizes a Stack

is produced, i.e. until no non-terminals remain in the line. The rules applied at each step in this example are the same ones whose numbers are shown in the tree of Figure 3.2.

Another algorithm for doing generation is given in Figure 3.5, and an example of its application is shown in Figure 3.6. There the stack is initialized with SENT. Then that is removed and replaced by NOUNPH, VERBPH, and a period, using rule 1. In the next iteration NOUNPH is removed and replaced by ADJ and NOUNPH, using rule 5. Then ADJ is removed and replaced by # and ADJP, using rule 9. Next, # is removed and put into the output stream, because it is a terminal symbol. The algorithm continues in this manner until the stack is empty, with the terminal symbols put into the output stream forming a string in the language. In the figure lines 48 through 53 were compressed to one line due to lack of space.

The essential difference between the two algorithms just presented is that the first one proceeds in a parallel fashion, expanding more than one non-terminal symbol at each step, whereas the second one is strictly serial. It can be seen by comparing the column of rule numbers in Figure 3.6 with the numbers on the tree in Figure 3.2 (and also from the output stream column) that the second algorithm "grows" the tree one branch at a time from left to right. The first algorithm may grow several branches at each step.

The serial approach of the second algorithm has some advantages for the computer. For example, less working storage

Line	Rule	Output	Stack
1			SENT
2	1		NOUNPH VERBPH .
3	5		ADJ NOUNPH VERBPH .
4	9		# ADJP NOUNPH VERBPH .
5		#	ADJP NOUNPH VERBPH .
6	24		T H E NOUNPH VERBPH .
7		T	H E NOUNPH VERBPH .
8		H	E NOUNPH VERBPH .
9		E	NOUNPH VERBPH .
10	6		NOUN VERBPH .
11	8		# NOUNP VERBPH .
12		#	NOUNP VERBPH .
13	21		M E N VERBPH .
14		M	E N VERBPH .
15		E	N VERBPH .
16		N	VERBPH .
17	2		VERBPH NOUNPH .
18	4		VERB NOUNPH .
19	7		# VERBP NOUNPH .
20		#	VERBP NOUNPH .
21	19		VERBS NOUNPH .
22	29		U N L O A D NOUNPH .
23		U	N L O A D NOUNPH .
24		N	L O A D NOUNPH .
25		L	O A D NOUNPH .
26		O	A D NOUNPH .
27		A	D NOUNPH .
28		D	NOUNPH .
29	5		ADJ NOUNPH .
30	9		# ADJP NOUNPH .
31		#	ADJP NOUNPH .
32	24		T H E NOUNPH .
33		T	H E NOUNPH .
34		H	E NOUNPH .
35		E	NOUNPH .
36	5		ADJ NOUNPH .
37	9		# ADJP NOUNPH .
38		#	ADJP NOUNPH .
39	25		B I G NOUNPH .
40		B	I G NOUNPH .
41		I	G NOUNPH .
42		G	NOUNPH .
43	6		NOUN .
44	8		# NOUNP .
45		#	NOUNP .
46	22		NOUNS S .
47	31		S H I P S .
48-53		SHIPS.	

Figure 3.6. An Example Application of the Stack Algorithm

is required at each step. Also, the left to right production of output makes it possible to overlap output and processing. This approach is more pleasing from a psycholinguistic point of view also, as it seems to more closely resemble the way people produce text. This algorithm forms the basis for the encoding procedure to be discussed shortly.

While there are many "good" sentences which can be produced by the grammar of Figure 3.1, using either of the algorithms of Figures 3.3 and 3.5, there are an infinite number of "bad" sentences which can be produced also. The following are a few of the strings which could be generated:

THE MEN UNLOAD THE BIG SHIPS.
THE BIG SHIPS ARE BEING UNLOADED.
BIG BOYS HAVE BEEN SERVICING THE BOAT.
BIG THE MENS HAVS BEED BOAT BOYS.
THE THE THE MAN ARE SHIP.

The source of the problem is, of course, the fact that there is no criterion stated for selecting among the possible rules to apply at each step of the generation process. This problem could be avoided by having only one rule for each non-terminal symbol, but this would result in needing many more symbols and many more rules (many of which would be almost identical) to be able to produce a particular set of strings. This approach would be better than the canned message approach, but not by much. (In the simplest canned message approach there is essentially one rule per message, and each rule has

only terminal symbols on the right, i.e. the characters of the message.) Increasing the number of symbols and rules in this fashion is unsatisfactory from a linguist's point of view because a grammar should capture as many generalities as possible in order to describe the language as simply as possible.

1(b) Work by Others

For a program that generated "grammatical nonsense" Yngve [47,48] used phrase structure rules that had subscripts on the non-terminal symbols to further limit the set of rules that had to be considered when making a selection at each step of the generation process. Using this scheme two symbols which differ only in their subscript (e.g. ADJ_0 and ADJ_1) could be considered to be either the same or to be different, depending upon where they are used. This makes it possible to easily control such things as having a "the" appear only at the beginning of a noun phrase. Yngve's program selected randomly from the set of permissable rules at each step and also selected randomly from appropriate sets of vocabulary items. The sentences thus produced were grammatically correct, but usually nonsensical (e.g."The dry beets are mixed of cash climate.").

Klein and Simmons [11] added a monitoring system to Yngve's random generation process to produce "coherent discourse." This was accomplished essentially by specifying relations among the vocabulary items, and then allowing only

those sentences which did not violate the relations to actually be printed (e.g. "Crops include tobacco."). Sentences like the example in the paragraph above would be discarded. Klein later expanded on this to automatically set up the necessary relations by processing an input text [12]. By using an appropriate control routine and another for inserting pronouns he was able to do "automatic paraphrasing" of the input paragraph. He also did some experiments in the "control of style" with this system by varying the probabilities of selection associated with the rules and the probability of selecting a pronoun versus a noun [13]. While this work is of theoretical interest, it is of limited usefulness in a practical application at this time.

2. Encoding Rules

The encoding rules of this system are basically phrase structure rules, and the encoding algorithm is basically the generation algorithm given in Figure 3.5. However, there are significant differences. Rather than simply names, the objects referred to in the rules are arbitrarily complex structures of attribute-value pairs (i.e. records). Also, within the rules procedures can be specified. These procedures are executed during encoding both to help determine the applicability of a rule and to build new structures when a rule is applied. By means of these procedures the rule writer has at his disposal the complete capability of the computer.

The basic idea of attaching procedures to phrase structure rules has been used for several years in syntax-directed com-

piling. A discussion of this other work will be deferred to the next chapter however, because it is more closely related to the decoding portion of this system. The use of this idea for generating text has not been seen in the literature. The language in which these procedures are written was developed specifically for this system and essentially consists of facilities for conveniently building and interrogating record structures.

Figure 3.7 shows a set of encoding rules for roughly the same portion of English that is described by the phrase structure rules of Figure 3.1. The encoding rules cut out the non-English, however. The similarities and differences between the two sets of rules may be readily noted, with the most obvious difference being the additional information given in parentheses. It can be seen that in some cases there are two or three encoding rules that are basically the same as some one phrase structure rule. For example, rules 4, 5, and 6 in Figure 3.7 are basically the same as rule 3 in Figure 3.1. It can also be seen that several of the vocabulary-type phrase structure rules do not have corresponding encoding rules.

The rules given in Figure 3.7 are grouped into two strata - lexology and morphology. As can be seen, the lexology deals with clauses and phrases, and the morphology deals with words and parts of words. This explicit stratification is arbitrary however, and is not required by the system.

Lexology for encoding English:

```
1    SENT(PASSIVE)  -->  NOUNPH(%GOAL(SENT))
                         VERBPH(%SENT,NUMB=NUMB(GOAL),-GOAL)  .
2    SENT  -->  NOUNPH(%AGENT(SENT))
                VERBPH(%SENT,NUMB=NUMB(AGENT),-AGENT)  .
3    VERBPH(GOAL)  -->  VERBPH(-GOAL)
                        NOUNPH(%GOAL(VERBPH))
4    VERBPH(PERFECT)  -->  VERB('HAV',VFORM=VFORM(VERBPH))
                           VERBPH(-PERFECT,-VFORM,PASTPART)
5    VERBPH(PROG)     -->  VERB('BE',VFORM=VFORM(VERBPH))
                           VERBPH(-PROG,-VFORM,PRESPART)
6    VERBPH(PASSIVE)  -->  VERB('BE',VFORM=VFORM(VERBPH))
                           VERBPH(-PASSIVE,-VFORM,PASTPART)
7    VERBPH  -->  VERB(%VERBPH)
8    NOUNPH(¬PRM)  -->  ADJ('THE')  NOUNPH(PRM)
9    NOUNPH(SIZE$'RELSIZ')  -->  ADJ(SUP=SIZE(NOUNPH))
                                 NOUNPH(-SIZE)
10   NOUNPH  -->  NOUN(%NOUNPH)
```

Morphology for encoding English:

```
11   VERB  -->  #  VERBP(%VERB,E*,EN*)
12   NOUN  -->  #  NOUNP(%NOUN)
13   ADJ   -->  #  ADJP(%ADJ)
14   VERBP('BE',SING)  -->  I  S
15   VERBP('BE',PLUR)  -->  A  R  E
16   VERBP('HAV',SING)  -->  H  A  S
17   VERBP(PRESPART)      -->  VERBS(SUP(VERBP))  I  N  G
18   VERBP(PASTPART,EN)   -->  VERBS(SUP(VERBP))  E  N
19   VERBP(PASTPART)      -->  VERBS(SUP(VERBP))  E  D
20   VERBP(E,SING)        -->  VERBS(SUP(VERBP))  E  S
21   VERBP(E,PLUR|INFIN)  -->  VERBS(SUP(VERBP))  E
22   VERBP(SING)          -->  VERBS(SUP(VERBP))  S
23   NOUNP('MAN',PLUR)  -->  M  E  N
24   NOUNP(PLUR)  -->  NOUNS(SUP(NOUNP))  S
```

Figure 3.7. A Set of English Encoding Rules

What appears in parentheses in the rules are the procedures to be executed during encoding. It is important to realize that in general what appears in parentheses are not simply the names of procedures, but rather the procedures themselves, written in the language developed for this system. The names appearing in these procedures are mostly those of indicators, attributes, and named records. The declarations for those names relevant to Figure 3.7 have been extracted from Appendices A and B and appear in Figure 3.8.

The parentheses on the left sides of rules are said to contain "condition specifications," and those on the right contain "creation specifications." These are the same as the creation specifications that are used in defining named records, as discussed in Section A.8 of Chapter II. As was seen there, the elements of creation specifications basically result in the setting of attribute and indicator values. The elements of condition specifications, on the other hand, basically test the values of attributes and indicators to produce a "true" or "false" outcome upon which a decision can be made to guide the processing. The primary function of these conditions is to select a rule to apply out of a group of rules with the same non-terminal symbol on the left, a problem which was discussed in the previous subsection.

2(a). Terminology

There is some terminology which must be defined at this point. A segment is a string of contiguous characters, including spaces and punctuation, in a text. A segment could

Indicators:

```
SUFX  1-10
   NSFX  1,   E  2,   S  3,   ES  4
   ING  5,   ED  6,   ER  7,   EN  8
TRANS  12
VFORM  13-18
   NUMB  14-15,   SING  14,   PLUR  15
   INFIN  16,   PRESPART  17,   PASTPART  18
PASSIVE  25,   PROG  26,   PERFECT  27,   PRM  33
```

Attributes:

```
SUP  1,   NAME  10
```

Named records:

```
ACTIVITY ('ACTION')
AGENT    ('ATTR')
ARRIV    ('EVENT',E,ES,ING,ED,ER)
BE       ('ATTRVERB',ING,EN)
BIG      ('RELSIZ')
BOAT     ('MOBENTY')
BOY      ('PERSON')
DET      (ATTRIB='DETERM')
EVENT    ('ACTION')
GOAL     ('ATTR')
HAV      ('ATTRVERB',E,ING,TRANS)
LOAD     ('ACTIVITY',NSFX,S,ING,ED,ER,TRANS)
MAN      ('PERSON')
RELSIZ   ('RELVAL',ATTRIB='SIZE')
SERVIC   ('ACTIVITY',E,ES,ING,ED,ER,TRANS)
SHIP     ('MOBENTY')
SIZE     ('ATTR')
THE      ('DET')
UNLOAD   ('ACTIVITY',NSFX,S,ING,ED,ER,TRANS)
```

Figure 3.8. Relevant Declarations from Appendices A and B

actually consist of no characters (i.e. the null string) or it could be several paragraphs long. A <u>segment type</u> is just a "type of segment" and is best defined by example. Sentence is a segment type, as are noun phrase, verb phrase, verb, verb part, verb stem, #, T, H, etc. In other words, a sentence is a particular type of segment, as is a noun phrase, a verb phrase, a T, an H, etc. The names used for segment types in the encoding rules (e.g. SENT, VERBPH, #, T) correspond to the non-terminal and terminal symbols in the phrase structure rules.

<u>Terminal segment types</u> are those for which there is only one possible string, i.e. the individual characters, such as each of the letters of the alphabet, the digits 0 through 9, and the special characters available in the EBCDIC code (e.g. the period, comma, question mark, and semicolon). The names of the terminal segment types are one character in length and are the same as the character of the segment they describe. For example, the segment type T describes the segment "t"; the string "th" consists of a T segment followed by an H segment. A special case is the segment type #, which describes a space in a string of text. Another special case is the null string, which will be mentioned later.

The names of <u>non-terminal segment types</u> (e.g. SENT, VERBPH) are more than one character in length and are made up by the user in exactly the same fashion as all other names in the system (i.e. names for attributes, indicators, and named records). A name must begin with a letter and may be

of any length, but only the first eight characters are kept by the system. It is possible to use the same name for both a segment type and for something else, such as a named record, as long as no ambiguity results where they appear. Of course, all names should be chosen in such a way as to make the task of writing and reading rules as easy as possible.

A <u>segment record</u> is a record in the cell structure that contains information about a segment. The kind of information depends on the type of segment. For example, at some point in the processing (although, not with the particular set of rules shown in Figure 3.7) a VERBPH segment record might contain attributes like SUBJECT and OBJECT and indicators like PASSIVE and PERFECT. A VERBP segment record would have an attribute to tell what the verb is and indicators to tell about its form. Terminal segment types would normally not have records associated with them because all of the information about the segment is contained in its name (e.g. a T is simply a "t").

<u>Encoding</u> is the process of converting segment <u>records</u> into segments of <u>text</u>. In general the encoding rules specify how higher-level segment records are to be converted to a series of lower-level segment records, each describing shorter, contiguous segments. At each level there are rules specifying conversion to still lower levels. At the lowest level output is actually produced. Six levels can readily be seen in the rules of Figure 3.7 - sentence, phrase, word, part, stem, and character. It is important to realize that encoding does <u>not</u> mean simply <u>rewriting</u> strings of characters, but rather it

involves converting information from one form to another (i.e. from a multi-dimensional network of attribute-value pairs to a linear string of characters in a text). This is one of the essential characteristics of Stratificational Grammar [18 , pg. 40].

Before going into the details of the language in which condition and creation specifications are written, the encoding algorithm will be presented and two example encodings will be given to motivate the discussion.

3. The Encoding Algorithm, with Examples

The encoding algorithm is shown in Figure 3.9. It can be seen that it is basically the same as the generation algorithm given in Figure 3.5. The significant difference is that this algorithm deals with segment records instead of just symbols. It must keep pointers to them on the stack, it must test conditions in them when determining which rule to apply, and it must create new ones when applying a rule. Also, this algorithm has two additional ways of producing output by using certain information in a segment record. This is stated in parts (b) and (ii) of the algorithm. The first of these will be discussed later, and the second, which is the way most output is produced, will be explained shortly in the discussion of the example encodings.

3(a). First Example

Figure 3.10 is an example of the application of the encoding algorithm of Figure 3.9 using the set of rules given

(1) Put a segment type name and a pointer to a segment record together on the stack to begin.

(2) Take the top segment type name and the associated segment record pointer (for a non-terminal segment type) off the stack, and examine the segment type:

   (a) if it is a terminal segment type, put its name into the output stream.

   (b) if it is the special segment type OUTPUT, perform the specified output operation according to the values of attributes 11-15 of the associated segment record.

   (c) otherwise, examine each rule that has this segment type on the left until either a rule is found for which the conditions specified in parentheses are met, or until the list of rules is exhausted:

      (i) if a rule is found, create segment records according to the specifications given in parentheses on the right side and put the segment type names, along with pointers to the associated segment records, onto the stack.

      (ii) otherwise, put into the output stream the name of the named record which is pointed to by attribute 1 (i.e. the SUP) of the segment record whose pointer was taken off the stack.

(3) Repeat step 2 until the stack is empty.

Figure 3.9. The Encoding Algorithm

Line	Rule	Output	Stack
1			SENT(R1)
2	2		NOUNPH(R2) VERBPH(R4) .
3	8		ADJ(R5) NOUNPH(R6) VERBPH(R4) .
4	13		# ADJP(R5) NOUNPH(R6) VERBPH(R4) .
5		#	ADJP(R5) NOUNPH(R6) VERBPH(R4) .
6		THE	NOUNPH(R6) VERBPH(R4) .
7	10		NOUN(R6) VERBPH(R4) .
8	12		# NOUNP(R6) VERBPH(R4) .
9		#	NOUNP(R6) VERBPH(R4) .
10	23		M E N VERBPH(R4) .
11		M	E N VERBPH(R4) .
12		E	N VERBPH(R4) .
13		N	VERBPH(R4) .
14	3		VERBPH(R7) NOUNPH(R3) .
15	7		VERB(R7) NOUNPH(R3) .
16	11		# VERBP(R7) NOUNPH(R3) .
17		#	VERBP(R7) NOUNPH(R3) .
18		UNLOAD	NOUNPH(R3) .
19	8		ADJ(R5) NOUNPH(R8) .
20	13		# ADJP(R5) NOUNPH(R8) .
21		#	ADJP(R5) NOUNPH(R8) .
22		THE	NOUNPH(R8) .
23	9		ADJ(R9) NOUNPH(R10) .
24	13		# ADJP(R9) NOUNPH(R10) .
25		#	ADJP(R9) NOUNPH(R10) .
26		BIG	NOUNPH(R10) .
27	10		NOUN(R10) .
28	12		# NOUNP(R10) .
29		#	NOUNP(R10) .
30	24		NOUNS(R11) S .
31		SHIP	S .
32		S	.
33		.	

<u>Records</u>:

R1	(SUP:'UNLOAD',AGENT:R2,GOAL:R3)
R2	(SUP:'MAN',PLUR)
R3	(SUP:'SHIP',SIZE:'BIG',PLUR)
R4	(SUP:'UNLOAD',GOAL:R3,PLUR)
R5	(SUP:'THE')
R6	(SUP:'MAN',PLUR,PRM)
R7	(SUP:'UNLOAD',PLUR)
R8	(SUP:'SHIP',SIZE:'BIG',PLUR,PRM)
R9	(SUP:'BIG')
R10	(SUP:'SHIP',PLUR,PRM)
R11	(SUP:'SHIP')

Figure 3.10. First Encoding Example

in Figure 3.7, supported by the declarations given in Figure 3.8. The upper part of the figure is similar to Figure 3.6, which showed the generation of the same sentence, but using the algorithm of Figure 3.5 and the grammar of Figure 3.1.

The additional information given in parentheses in the stack represents the segment record pointers, with the contents of the segment records being given in the lower part of the figure. Each record is represented there by a collection of items separated by commas. Each item is either an attribute name and a value with a colon between them, or an indicator name. In this example each of the attribute values is a pointer, either to another segment record or to a named record, but of course in general they could also be numeric values or character string values as discussed in Chapter II. The appearance of an indicator name in one of these records means that that indicator is "on" in that record, i.e. the appropriate bit position of attribute 2 holds a 1 rather than a 0. (The bit positions corresponding to the various indicator names can be found in the declarations in Figure 3.8.)

The example of Figure 3.10 assumes that records R1, R2, and R3 are available to begin with. These records contain the information which is to be put out in the form of a sentence. It will be seen in a later section of this chapter that in producing the English description of a queuing problem, rules in the encoding semology essentially extract records like these from the Internal Problem Description and pass them to the lexology to output a sentence. The fact that these records

conform to the conventions of the IPD as described in Chapter II should be obvious. R1 is an action record, and R2 and R3 are entity records.

To begin the encoding algorithm SENT and a pointer to R1 are put on the stack. Then SENT and the pointer are taken off the stack. SENT is examined, and it fails tests (a) and (b), so the rules with SENT on the left must be examined according to (c). Rule 1 is looked at first, but it can not be applied because record R1 does not have the PASSIVE indicator "on." An indicator name appearing as a condition element in a rule specifies that that indicator of the current segment record must be "on" to result in an outcome of "true." When preceded by the "not" symbol, as in rule 8, it specifies that the indicator must be "off" to result in an outcome of "true."

Rule 2 is then looked at, and is applied because it has no conditions to be satisfied. Application of this rule results in the stack configuration shown in line 2 of Figure 3.10. Rule 2 says that a SENT "yields" a NOUNPH which is a copy of the AGENT of the SENT, followed by a VERBPH which is a copy of the SENT (but with its NUMB indicator set to equal the NUMB indicator of its AGENT, and then with its AGENT attribute erased), followed by a period.

When the name of the segment type on the left of the rule being applied appears in a creation specification, it refers to the segment record whose pointer was just taken off the stack. Therefore, when rule 2 was applied at this point, SENT referred to R1. The percent sign (%) means to copy the record

designated by the name following it. Usually the copy forms the basis for the new segment record being created, as in both cases here. A left parenthesis can usually be read as "of." For example, AGENT(SENT) is "the AGENT of the SENT," i.e. the AGENT attribute of the SENT record, where the SENT record is R1 in this case. A minus sign (-) in front of an attribute name means to erase that attribute from the record. A minus sign in front of an indicator name means to turn that indicator "off," i.e. set the bit positions associated with the name to 0.

Just to preserve space in the figure, rather than making a copy of a record which would result in having two identical records, the original is used. Therefore, the segment record pointer associated with the NOUNPH in the stack points to R2, which is what the AGENT of the SENT pointed to. The segment record pointer associated with the VERBPH in the stack points to R4, which is a copy of R1 but without the AGENT attribute and with the PLUR indicator, as dictated by the creation specification associated with the VERBPH in rule 2.

It may not be obvious how the PLUR indicator got set in R4. In the declarations shown in Figure 3.8 it can be seen that SING is a name for bit position 14, PLUR is a name for position 15, and NUMB is a name for both positions 14 and 15 together. In Figure 3.10 record R2 is shown as having only the PLUR indicator "on," which means bit position 15 (PLUR) is 1 and all others are 0. When the creation element NUMB=NUMB(AGENT) was executed during the application of rule 2, bits 14 and 15 of record R4 (the record being built) were

set equal to bits 14 and 15 of record R2 (the record pointed to by the AGENT attribute of this record). This resulted in bit 15 of R4 being a 1, which is shown as PLUR in Figure 3.10. If R2 had the SING indicator (bit 14) "on," then SING would have been set in R4.

The effect of applying rule 2 is to take the information about the whole sentence segment and split it into two separate pieces of information about two contiguous segments, the subject and the predicate, each of which can be expanded independently. It also tacks the period, which is actually a third contiguous segment, onto the end. Information about the number of the subject is given to the predicate so that the proper form of the verb can be put out eventually. Rule 1 would do essentially the same thing if it were applied, except that because it is for a passive sentence the GOAL, rather than the AGENT, would be made the subject.

To obtain line 3 of the example, NOUNPH and the pointer to R2 are taken off the stack. NOUNPH is examined, and it fails tests (a) and (b), so the rules with NOUNPH on the left must be examined according to (c). Rule 8 is looked at first and is applied because the condition is met (i.e. R2 does not have its PRM indicator "on"). Application of this rule results in an ADJ with a SUP of 'THE' (record R5) and a new NOUNPH (record R6) being put onto the stack, as can be seen in line 3. This new NOUNPH is a copy of the old NOUNPH (i.e. R2, the one just taken off the stack) but with the PRM indicator set.

The explanation of a few more conventions is required here. When the name of a named record appears in single quotes as a creation element (e.g. 'THE'), it means to set attribute 1 (i.e. the SUP) of the record being built to point to that named record. When a segment type on the right of a rule has the same name as the segment type on the left (NOUNPH in this case), the new segment record created is automatically made to be a copy of the old one (i.e. the one just taken off the stack). This copy may then be modified by the creation elements. When the name of an indicator appears as a creation element (e.g. PRM), it means to turn that indicator on in the record being built.

The effect of applying rule 8 is to eventually put out a "the" at the beginning of a noun phrase. This is done by creating an ADJ segment record for "the", followed by a NOUNPH segment record with all of the old information plus one additional piece of information. This additional information is the PRM indicator, which will insure that another "the" does not get put out at the beginning of this phrase.

The next line in the example involves applying rule 13, and then in the following line the terminal segment type # comes off the stack and its name is put into the output stream according to part (a) of the algorithm. Then ADJP and its associated pointer to R5 are taken off the stack. ADJP fails tests (a) and (b) and there are no rules given with ADJP on the left, so part (ii) of the algorithm is applied. The name

of the named record pointed to by attribute 1 (i.e. the SUP) of record R5 which was just taken off the stack is "the", so the characters "t", "h", and "e" are put into the output stream according to part (ii).

The output shown in lines 18, 22, 26, and 31 is produced in exactly the same manner as that of line 6, i.e. by applying part (ii) of the algorithm. Most of the output produced using this system comes about in this way. In cases like "unload", "big", and "ship" it works out nicely because the names chosen for most of the named records in the concept structure are the stems of the English words that are used to refer to those concepts. In cases like "the", it is simply very convenient to write such things as ADJ('THE') in a rule. The English encoding morphology then need only handle word endings and irregularities, as can be seen in Figure 3.7.

After some additional discussion about the set of rules, including further explanation of the conventions used in condition and creation specifications, the reader should be able to trace through the remainder of the example given in Figure 3.10 without a step-by-step description.

3(b). <u>The Other Rules</u>

Rule 3 takes care of putting a direct object into the sentence when there is to be one. It would be applied when a VERBPH has a GOAL attribute. This rule is typical of many of the encoding rules in the set used for describing queuing problems to be discussed in a later section. This kind of

rule results in a copy of the original segment record, but without the feature that made the rule apply (the GOAL attribute in this case). In addition another segment record is created with information related to that feature (the NOUNPH which is a copy of the GOAL in this case). Rules 4, 5, 6, 8, and 9 are similar in this respect.

Rules 4, 5, and 6 take care of putting out the appropriate auxiliary verbs for the perfect, progressive, and passive. As can be seen in the declarations of Figure 3.8, the verb form indicator VFORM "covers" bit positions 13-18 which includes the indicators SING, PLUR, INFINitive, PRESPART (present participle), and PASTPART (past participle).. The creation element VFORM=VFORM(VERBPH) which appears in all three rules serves to pass the necessary verb form information down to the morphology. (This works similarly to NUMB, which was described earlier.) Rule 7 would apply to a VERBPH after the auxiliaries were taken care of and simply passes the main verb down.

Rule 9 asks if the SIZE attribute of a NOUNPH is "in the set" 'RELSIZ'. If it is, the rule yields on ADJ with its SUP made to point to the same named record that the SIZE attribute of the NOUNPH is pointing to, followed by a NOUNPH without a SIZE attribute. In the example this rule is applied to obtain line 23. Record R8 has a SIZE attribute pointing to the named record 'BIG'. In Figure 3.8, it can be seen that 'BIG' has a SUP of 'RELSIZ', which means that 'BIG' <u>is</u> in the set 'RELSIZ'. If record R8 had no SIZE attribute, or if its SIZE attribute were pointing to a record representing something like "1000 tons",

which would not be in the set 'RELSIZ', then rule 9 would not be applied. Rule 10 simply passes the "head" of the NOUNPH down.

Rules 11, 12, and 13 primarily insert the space at the beginning of a word. In rule 11 however, the values of two indicators (E and EN) are also copied into the VERBP segment record from the named record pointed to from attribute 1 (i.e. the SUP). The stars after the indicator names in the creation specification mean to do that. For example, if a VERB with a SUP of 'SERVIC' came through rule 11, it would result in a VERBP with its E indicator "on" because that indicator is "on" in the named record 'SERVIC', as can be seen in Figure 3.8.

Rules 14, 15, 16, and 23 handle certain irregular verbs and an irregular noun. When the name of a named record appears in single quotes as a condition element, an outcome of "true" will result if attribute 1 (i.e. the SUP) is a pointer to that named record. A comma between elements in a condition specification designates an "and" relationship between the elements, and a vertical bar such as in rule 21 designates an "or" relationship. Rule 14, for example, says that a VERBP with a SUP of 'BE' and the SING indicator "on" yields an I followed by an S. In other words the singular (third person, present tense) form of "be" is "is".

Rules 17 — 22 and 24 all have to do with attaching an appropriate suffix to a stem which needs one. When the name of an attribute of some other record appears as a creation

specification, it means to set the value of that attribute in this new record equal to its value in the other record. For example, SUP(VERBP) is equivalent to SUP=SUP(VERBP). Rule 21, then, says that a VERBP with both an E indicator and either a PLUR or INFINitive indicator "on" yields a VERBS with the same SUP as the VERBP, followed by an E. The SUP in this case would be a pointer to a named record such as 'SERVIC', 'LEAV', or 'HAV'. The name of the SUP (e.g. "servic") would be put into the output stream when the VERBS comes off the stack, according to part (ii) of the encoding algorithm, and then an "e" would be put into the output stream when the E comes off the stack, according to part (a), thus forming the complete word.

At this point the reader may find it beneficial to trace through the remainder of the example given in Figure 3.10 before going on to the second example.

3(c). <u>Second Example</u>

A second example application of the encoding algorithm is given in Figure 3.11, which is in the same format as Figure 3.10. The purpose of this example is to show how the same set of rules can result in the production of a sentence of quite different appearance by starting with just slightly different information. The example also serves to illustrate the encoding of a verb phrase which includes auxiliaries.

Just as in the previous example this one assumes that records R1, R2, and R3 are available to begin with. R2 and

Line	Rule	Output	Stack
1			SENT(R1)
2	1		NOUNPH(R3) VERBPH(R4) .
3	8		ADJ(R5) NOUNPH(R6) VERBPH(R4) .
4-14		#THE#BIG#	
15		SHIP	S VERBPH(R4) .
16		S	VERBPH(R4) .
17	5		VERB(R10) VERBPH(R11) .
18	11		# VERBP(R10) VERBPH(R11) .
19		#	VERBP(R10) VERBPH(R11) .
20	15		A R E VERBPH(R11) .
21		A	R E VERBPH(R11) .
22		R	E VERBPH(R11) .
23		E	VERBPH(R11) .
24	6		VERB(R12) VERBPH(R13) .
25	11		# VERBP(R12) VERBPH(R13) .
26		#	VERBP(R12) VERBPH(R13) .
27	17		VERBS(R14) I N G VERBPH(R13) .
28		BE	I N G VERBPH(R13) .
29		I	N G VERBPH(R13) .
30		N	G VERBPH(R13) .
31		G	VERBPH(R13) .
32	7		VERB(R13) .
33	11		# VERBP(R13) .
34		#	VERBP(R13) .
35	19		VERBS(R15) E D .
36		UNLOAD	E D .
37		E	D .
38		D	.
39		.	

<u>Records</u>:

R1	(SUP:'UNLOAD',AGENT:R2,GOAL:R3,PASSIVE,PROG)
R2	(SUP:'MAN',PLUR)
R3	(SUP:'SHIP',SIZE:'BIG',PLUR)
R4	(SUP:'UNLOAD',AGENT:R2,PLUR,PASSIVE,PROG)
R5	(SUP:'THE')
R6	(SUP:'SHIP',SIZE:'BIG',PLUR,PRM)
R7	(SUP:'BIG')
R8	(SUP:'SHIP',PLUR,PRM)
R9	(SUP:'SHIP')
R10	(SUP:'BE',PLUR)
R11	(SUP:'UNLOAD',AGENT:R2,PASSIVE,PRESPART)
R12	(SUP:'BE',PRESPART)
R13	(SUP:'UNLOAD',AGENT:R2,PASTPART)
R14	(SUP:'BE')
R15	(SUP:'UNLOAD')

Figure 3.11. Second Encoding Example

R3 are the same as in the earlier figure, but R1 has had two indicators added to it - PASSIVE and PROGressive.

The details of lines 4 through 14 have been omitted from Figure 3.11 due to a lack of space. All that is shown for those lines is a summary of the output which they would produce, which can be seen to be the same as that of lines 20 through 30 of Figure 3.10. The item on the top of the stack in line 2 of the second example is exactly the same as the item on the top of the stack in line 18 of the first example, i.e. NOUNPH(R3). It follows then that the expansion of that item would be identical in both cases, going through exactly the same steps and producing exactly the same output, i.e. "the big ships". All that would be different in the two figures are some of the segment record numbers, because more of them had already been used at this point in the processing of the first example. The 15 records shown in Figure 3.11 include those referenced from the 11 lines that were omitted.

In this example when SENT(R1) is taken off the stack, rule 1 applies because the PASSIVE indicator in record R1 is "on". Then the NOUNPH for the subject is expanded as already discussed, and the VERBPH is expanded. This latter expansion first produces a "be -ing" for the progressive, followed by a "be -en" for the passive, and finally the main verb. As will be seen in the section on English encoding rules for queuing problem descriptions, not much more is required in the way of rule complexity to be able to handle interrogatives, negatives, and modals.

The rules given in Figure 3.7 ignore the agent of an action in the passive construction. However, only the following two rules would need to be added to that set to produce an agentive-by phrase:

```
VERBPH(AGENT)  -->  VERBPH(-AGENT)
                    PREP('BY')
                    NOUNPH(%AGENT(VERBPH))

PREP  -->  #  PREPP(%PREP)
```

The first rule says essentially that if a VERBPH has an AGENT, put it out after the rest of the VERBPH as the object of a prepositional phrase with the preposition "by". The other rule is similar to rules 11 — 13.

If the second example were done again with these rules included, the sentence which would be put out is, "The big ships are being unloaded by the men." These rules would have no effect on the first example, because the VERBPH segment record there did not have an AGENT attribute.

3(d). Segment-Type Records

In the computer, segment type names do not actually appear in the stack. Rather, what are there are pointers to segment-type records. There is a segment-type record for each segment type that appears in a set of rules. This record has a NAME attribute with the name of the segment type and another attribute with a pointer to the first encoding rule that has that segment type on the left. Each rule then has a pointer to the next rule that has the same segment type on the left,

effectively forming a list of rules for that segment type. Segment-type records are created by the system when it compiles a set of rules into their internal format. More will be said about this internal format later.

4. The Language of Specification Elements

It was stated earlier that what appears in parentheses in the rules are procedures to be executed during encoding. As could be inferred from the discussion of the examples just presented, each of these procedures consists of one or more steps, called elements, separated by commas, or in some cases separated by vertical bars. A number of different kinds of elements were described there. It was seen that the elements in creation specifications place information in records to be used later, and that the elements in condition specifications evaluate to "true" or "false", using some of this information, to guide the processing.

Although it may appear that there are a "hodge-podge" of specification elements, that really is not the case. All creation elements are basically of the form:

```
        attribute of record  =  value
    or  indicator of record  =  value
```

and all condition elements are basically of the form:

```
        value  .relation.  value
```

However, there are several notational conventions available in order to make the language more convenient to use. (These

are what make it appear as somewhat of a "hodge-podge" at first.)

Most of the creation and condition elements in the rules given in Figure 3.7 employ notational conventions, i.e. they are written in a "shorthand" fashion. For example, the condition specification in rule 1 could have been written PASSIVE.EQ.1 or PASSIVE.NE.0, instead of simply PASSIVE. That of rule 3 could be GOAL.NE.0, instead of simply GOAL, and that of rule 14 could be SUP.EQ.'BE',SING.NE.0, instead of 'BE',SING. The creation specifications in rule 8, for example, could have been written SUP='THE' and PRM=1, instead of simply 'THE' and PRM. The E* of rule 11 could be E=E(SUP). Even the copying could be specified this way instead of using %, but it would require a creation element for each attribute in the record. Clearly the notational conventions offer some conveniences to the user. They also increase the efficiency with which the system can compile and execute these procedures.

It is the purpose of this section to describe in detail the language of specification elements, a large part of which has to do with the notational conventions. The information presented here is relevant not only to encoding rules, but also to decoding rules and to named record definitions, because the same routines in the FORTRAN program underlying this system are used to process the specifications given in parentheses for all three.

Figure 3.12 outlines the information to be presented in this section. The order of the two is slightly different

```
1.  Basic forms of creation elements
      attribute-number(pointer-value) = value      an(pv)=v
      indicator-number(pointer-value) = value      in(pv)=v

2.  Basic form of condition elements
      value..relation. value    v.r.v

3.  Kinds of relations (r)
      EQ, NE, LT, LE, GT, GE

4.  Attribute designator form
      attribute-number(pointer-value)     an(pv)

5.  Indicator designator form
      indicator-number(pointer-value)     in(pv)

6.  Kinds of attribute numbers (an)
      explicit numeric (e.g. @14)
      explicit symbolic (e.g. SUP)
      implicit symbolic (e.g. AGENT)
      indirect (e.g. @N)

7.  Kinds of indicator numbers (in)
      explicit symbolic (e.g. SING)

8.  Kinds of values (v)
      pointer value (pv)
      number value (nv)
      string value (sv)

9.  Kinds of pointer values (pv)
      SEGMENT, SEG, RECORD
      MEMORY, MEM
      'named-record-name'
      segment type name
      attribute designator for a type-3 cell
      = /%pv
      an(/$n(pv)

10. Kinds of number values (nv)
      simple number value (snv)
      snv+snv, snv-snv, snv*snv, snv/snv

11. Kinds of simple number values (snv)
      integer constant (decimal, hex, octal, binary)
      attribute designator for a type-0 cell
      indicator designator

12. Kinds of string values (sv)
      "string-of-EBCDIC-characters"
      attribute designator for a type-1 or type-2 cell
```

Figure 3.12. The Language of Specification Elements

however, because the figure is intended to be used primarily for reference. The material to be discussed here has a recursive nature to it which makes it difficult to find a natural starting point for the discussion.

4(a). Attribute, Record and Value

There are basically three kinds of values which this system deals with - pointer values, number values, and string values. A pointer value is the identification number of some record in the cell structure, i.e. it points to some record. An attribute that has a pointer value would be stored as a type-3 cell in the cell structure. A number value is simply an integer and is usually, but not always, the count of something. An attribute with a number value would be stored as a type-0 cell. A string value is a string of EBCDIC characters and often is the name of something. An attribute with such a value requires either a type-1 or a type-2 cell, depending upon the length of the string. The various types of cells were discussed in detail in Section A of Chapter II.

In order to make a statement of the form "attribute of record = value" meaningful, there are three distinct pieces of information which must be given - what attribute, what record, and what value. At the lowest level in this system the attribute is specified strictly by number, as discussed in the previous chapter, although at a higher level names may be associated with these numbers. The record is specified by a record identification number (i.e. the subscript of the

first cell of the record in the CELL array in the FORTRAN program). A pointer value, as defined above, identifies a record. The value part of the statement could be any value of any of the three kinds just defined.

Execution of a statement of the above form results in a cell being inserted in a record. The number (0, 1, 2, or 3) placed in the TYPE field of the cell would depend on the kind of value; the number placed in the ATTR field would be the specified attribute number; the ADDR field would depend on the kind of value, as TYPE does; and the LINK field would depend on what record this cell was being inserted into.

From the above discussion, the statement

attribute of record = value

could be written as

attribute(record) = value

or more specifically

attribute number (pointer value) = value

which could be abbreviated to

an(pv) = v

This is the basic form of all creation elements for giving a value to an attribute. Similarly,

in(pv) = v

where "in" stands for "indicator number", is the basic form for all creation elements that give a value to an indicator. Also,

v.r.v

is the basic form for all condition elements, where "r" stands for "relation".

4(b). Attribute and Indicator Designators

The form an(pv) may be called an attribute designator, where "an" specifies the attribute number in one of four ways to be discussed shortly, and "pv" specifies the record, i.e. it furnishes a record identification number. Similarly, the form in(pv) may be called an indicator designator, where "in" specifies the indicator number, as will be discussed, and "pv" specifies the record.

The attribute number (an) of an attribute designator can be given in any one of four ways - explicit numeric, explicit symbolic, implicit symbolic, and indirect. An explicit numeric specification consists of the symbol @ followed by a number which is the attribute number. For example, @15 means attribute number 15, i.e. a cell that has an ATTR field with the number 15 in it, as described in Chapter II. Similarly, @101 means attribute number 101. An explicit symbolic specification is simply a name which was declared to be an attribute name, as those shown in Appendix B, and the corresponding attribute number is as declared. The most widely used of these in the current sets of rules is SUP, which was declared as attribute 1. Because of that declaration, "SUP" and "@1" could be used interchangeably.

An implicit symbolic specification is the name of a named record used as an attribute name. The corresponding attribute number is the identification number of the named record, as discussed in Section A.6 of Chapter II, and need never be known explicitly by the user. AGENT and GOAL are examples of this.

An <u>indirect</u> specification consists of the symbol @ followed by another attribute designator which may itself have an attribute number of any of the three kinds discussed so far. In this case the attribute number being specified is actually the <u>value</u> (not the <u>number</u>) of the attribute designated after the @. For example, the pair of creation elements N=15,@N=21 would be equivalent to @15=21. Similarly, MTR='AGENT', @MTR='REC21' would be equivalent to AGENT='REC21'. Of course, in normal usage the two elements in either of the pairs just shown would not be adjacent, because in that case there would be no need for using indirect specification.

Indirect specification is especially useful for building lists, like the action list or the X-Y pairs in a function entity. For example, the LC attribute of some segment record may be initialized to 10 somewhere. Then to add a pointer, say pointing to record 'XYZ', to a list being maintained within the segment record, the pair of creation elements LC=LC+1, @LC='XYZ' could be used. Something similar may also be done within a "loop", possibly to examine each record on a list. Examples of this will be pointed out in the section discussing the various sets of rules that have been written for the queuing problem application.

It might be helpful to summarize and contrast the four ways of specifying the attribute number in an attribute designator. In the <u>explicit numeric</u> case the user knows exactly what attribute number he wants the system to use for holding some piece of information, and he wants to refer to it by that number. In the <u>explicit symbolic</u> case also, he knows exactly what attribute number he wants the system to use, but he wants to be able to refer

to it by a name (which he must declare). In the <u>implicit symbolic</u> case he does not care what attribute number the system actually uses for holding some piece of information, because he wants to refer to it only by name. (In this case the system uses the identification number of the named record that has that name.) In the <u>indirect</u> case, at the time of rule writing the user does not know exactly which attribute number he wants the system to use at some point in the processing, but he does know the name (or number) of another attribute which will contain the number as its value at that point, such as LC in the example just given. This is similar in concept to using a variable for a subscript in a computer program.

Nested indirect specification is not allowed (i.e. indirect specification cannot be used within the attribute designator that goes with the @), but this does not limit the capability of the language. The "trick" for achieving the same effect is essentially the same as that for doing subscripted subscripting in a programming language that allows subscripts of only a very limited form (i.e. compute the subscript as a separate step and store it in a simple variable which can then be used as the subscript).

The <u>indicator number</u> (in) of an indicator designator can be specified in only one way - explicit symbolic. Similarly to that for attribute numbers, an <u>explicit symbolic</u> specification for an indicator number is simply a name which was declared to be an indicator name, as those shown in Appendix B, and the corresponding indicator number is as declared.

In general, an indicator number is really a pair of numbers specifying a range of bit positions. In most cases however, both numbers of the pair are the same. In the examples given earlier SING was declared as a name for indicator 14 (i.e. the 14th bit from the right in the value of attribute 2 of a record), and NUMB was declared as a name for the two bits of indicators 14 and 15 together.

4(c). Pointer Values

There are several different items that may serve as pointer values (pv) in condition and creation specifications. First to be described here are five literal names that have special meaning. SEGMENT, SEG, and RECORD are three of these names. They are completely equivalent and can be used to refer to the "current" record, i.e. the record in which conditions are being tested or which is being built when the name is encountered. The value associated with these names is the identification number of this record. Because of a notational convention to be described later, these names are not needed very often. MEMORY and MEM are two names which are completely equivalent and are used to refer to the MEMORY record as discussed in Chapter II. The value associated with these names is the identification number of the MEMORY record (currently 1).

A named record name in single quotes, e.g. 'BE' or 'THE', may serve as a pointer value, with the value being the identification number of the named record. This is used for referring to the named record from any specification element. A

segment type name may serve as a pointer value also, but only in a creation specification, with the value being the identification number of the segment record "on the left of the rule" (i.e. the record whose pointer was just taken off the stack). A segment type name may not be used to refer to the current record. Examples of both of these kinds may be seen in the rules in Figure 3.7.

MEMORY, MEM, and named record names may be considered to be global variables whose values are established by the system and do not change during execution (i.e. during encoding and decoding). SEGMENT, SEG, RECORD and segment type names (when used as pointer values) may be thought of as variables maintained by the system too, but whose values change according to which rule and which record are being processed at the time.

An attribute designator can serve as a pointer value also, if the value of the designated attribute is the identification number of some record, i.e. if the attribute is stored as a type-3 cell. This introduces recursion into the definition of attribute designator, because a pointer value is a part of an attribute designator.

Some examples of valid specification elements involving pointer values may now be given. PTR(MEM)=RECORD would store the identification number of the current record being processed as the value of the PTR attribute of the MEMORY record. PTR('XYZ')='ABC' would store the identification number of the named record 'ABC' as the value of the PTR attribute of the named record 'XYZ'. If this were followed by MN(PTR('XYZ'))='ST'

which can be read as "MN of PTR of XYZ equals ST," this one would store the identification number of 'ST' as the value of the MN attribute of 'ABC'. Of course, MN('ABC')='ST' would do the same thing.

In all of the above examples the standard form an(pv)=v could be seen. In the most complex example given there the pv part was an attribute designator which itself had the form an(pv), i.e. PTR('XYZ'). There is no limit to the nesting of this sort which can be done in this language. In other words, an element of the form

```
      an(an(an(an(an(pv))))) = v
or    in(an(an(an(an(pv))))) = v
```

is perfectly acceptable. Actually, each of the attribute numbers (an) could be specified in any of the four ways described earlier, too, including indirect. (This is not the same as nested indirection, which involves more than one indirect @ within the same an.) The following could be a valid specification element:

```
PTR(@14(@MN(@101(SEG))(@PTR('XYZ')('DEF')))) = 1
an ( an (@an( an ( pv ))(@ an (  pv  )(  pv  )))) = v
an ( an (@an(    pv    )(     an     (  pv  )))) = v
an ( an (       an     (      an     (  pv  )))) = v
```

There are two more sources of pointer values yet to be described, each of which results in a pointer value only when appearing in a specific context. The first of these is %pv when it appears immediately after an equal sign. This form means to make a copy of the record designated by the pv and

consider the identification number of this new record to be the value of %pv. For example, the creation element MN('ABC') = %'ST' would store as the value of MN of 'ABC' the idenficiation number of a newly created copy of the named record 'ST'. The pv after the % can be any of the pointer values discussed.

The final pointer value to be described is $n(pv) when it appears immediately after an attribute designator which is followed by a left parenthesis. An example of its use could be PS(SEG)=PS(<u>$SUP(SEG)</u>), which essentially says to set the PS attribute of the segment record being built to the same value as the PS attribute of some record in the SUP chain of the record being built. In this example the pointer value associated with the underlined part is the identification number of the first record in the SUP chain of the current record that has a PS attribute, the attribute whose name is given just before the "($". If there were no PS attribute in any record in the chain, the pointer value associated with the underlined part would be the identification number of the last record in the chain (i.e. the first one with no SUP). Of course in this latter case PS(SEG) would turn out to be 0. The n in the form shown above can be the number of an attribute (e.g. 1) or the name of an attribute (i.e. either explicit or implicit symbolic). If neither is given, a 1 is assumed. (The name SUP was not really needed in the example given above, then.) The pv shown there can be any of the pointer values discussed.

The form of pointer value described in the above paragraph can be very convenient, and indeed is used a number of places in both the encoding and decoding rules written for the queuing problem application, as will be seen later. It makes it possible to easily access information stored an indeterminant number of levels away. Information may be stored this way in an effort to avoid duplication when several members of some set have the same value for some attribute. This was discussed in the previous chapter both for the PS attribute in the concept structure and for attributes like CONSUMPtion and CAPACITY in the Internal Problem Description.

4(d). Number and String Values

As mentioned earlier there are basically three kinds of values (v) which this system deals with. Any one of them can appear in place of the v in the forms given, i.e. an(pv)=v, in(pv)=v, and v.r.v. (It is unlikely however that any kind other than a number value would appear for the v in the form in(pv)=v.) Pointer values (pv) have been discussed first here because of the additional role they play in attribute and indicator designators. The other two kinds - number and string - will be discussed now.

A number value (nv) may be either a simple number value (snv) or an expression consisting of two simple number values separated by an arithmetic operator, i.e. snv+snv, snv-snv, snv*snv, or snv/snv. A simple number value may be either an integer constant, an attribute designator which designates an

attribute that is stored as a type-0 cell, or an indicator designator. An integer constant may be given in decimal, hexadecimal, octal, or binary notation, with a letter on the end specifying which one (none for decimal, Z for hexadecimal, K for octal, and B for binary). For example, the constants 26, 1AZ, 32K, and 11010B are all equivalent.

A string value (sv) may be either an EBCDIC constant, i.e. a string of characters within double quotes, or an attribute designator which designates an attribute that is stored as a type-1 or type-2 cell. When a string value appears in a condition element, i.e. sv.r.sv, only the first eight characters are considered in making the comparison. For example, the strings "ABCDEFGHIJ" and "ABCDEFGHXY" would compare as equal.

When an indicator designator is used as a value, the value is made up of the bits of the indicator right justified. When a value is stored in an indicator that has b bits in its range, the rightmost b bits of the value are stored. For example, if NEG is defined as a one-bit indicator, the element NEG=NEG+1 would serve to invert its value. (If NEG were 0, the rightmost bit of the value formed by the addition would be 1. If NEG were 1, the rightmost bit of the value formed by the addition would be 0.)

Although the general form for a condition element has been given as v.r.v, there is a restriction on the v which has not been stated, that is that the arithmetic expression kind of number value cannot be used. In other words, the v of v.r.v

can only be a pv, sv, or snv, all as defined earlier. The usual six relations are available for condition elements - equal (EQ), not equal (NE), less than (LT), less than or equal (LE), greater than (GT), and greater than or equal (GE). When the two values being compared are not of the same kind (e.g. a pv and an snv), they are considered to be "not equal". As stated above, when two sv's are compared, only the first eight characters of each are considered in the comparison.

Now that attribute and indicator designators and the various kinds of values have been discussed in detail, the notational conventions available in creation and condition elements may be described.

4(e). Notational Conventions

Probably the most useful notational convention is that the pv part of an attribute or indicator designator need not be stated when the attribute or indicator being referred to is in the current record (i.e. the record in which conditions are being tested or which is being built), as long as no ambiguity will result. The pv part which is omitted would be one of the literals SEGMENT, SEG, or RECORD. Many examples of this convention have already been seen, e.g. NEG=NEG+1 could have been written NEG(SEG)=NEG(SEG)+1. For another example, PTR(MN)='XY' is equivalent to PTR(MN(SEG))='XY'.

One place where this convention often cannot be used is in the indirect specification of an attribute number. An example of the use of indirect specification given earlier was

N=15,@N=21, the second element of which could have been written @N(SEG)(SEG)=21. However, if the attribute 15 of interest were in record 'XYZ' and the attribute N were still in the current record, the second element would have to be written @N(SEG)('XYZ')=21, which is equivalent to writing @15('XYZ')=21 (as long as attribute N has the value 15). If the pv SEG were omitted in this case, i.e. @N('XYZ')=21, this would be equivalent to @N('XYZ')(SEG)=21, which would not be appropriate unless attribute N were in 'XYZ' and the attribute 15 of interest were in the current record. If both attributes were in 'XYZ', then @N('XYZ')('XYZ')=21 would be appropriate. This situation occurs a number of times in both the encoding and decoding rules that have been written for the queuing problem application, as will be seen later. Leaving off the second pv in this last mentioned situation is an error which seems to be commonly made when writing rules in this language.

A tabulation of 17 notational conventions that can be used in writing specification elements appears in Figure 3.13. The first seven are for creation elements, and the last ten are for condition elements. About half of each are for attributes and the others are for indicators. The figure shows the form of the convention in the left hand column, the "full-blown" equivalent form in the right hand column, and examples in the middle column. For each convention only one example is shown, but in both forms. Many more examples of each can be found in the sets of rules in the appendices, although the form is sometimes hard to recognize there because of nested attribute

<table>
<tr><th></th><th></th><th>Form</th><th>Example</th><th>Equivalent Form</th></tr>
<tr><td rowspan="4">attributes</td><td>1.</td><td>an(pv)</td><td>SUP(VERBP)
SUP(SEG)=SUP(VERBP)</td><td>an(SEG)=an(pv)</td></tr>
<tr><td>2.</td><td>-an [(pv)]</td><td>-AGENT
AGENT(SEG)=0</td><td>an(pv)=0</td></tr>
<tr><td>3.</td><td>'nrn'</td><td>'BE'
@1(SEG)='BE'</td><td>@1(SEG)='nrn'</td></tr>
<tr><td>4.</td><td>%pv</td><td>%SENT
@1(SEG)=@1(SENT)
@2(SEG)=....,etc.</td><td>@1(SEG)=@1(pv),
@2(SEG)=@2(pv),etc.</td></tr>
<tr><td rowspan="3">indicators</td><td>5.</td><td>in [(pv)]</td><td>PRESPART
PRESPART(SEG)=1</td><td>in(pv)=1111...111B</td></tr>
<tr><td>6.</td><td>-in [(pv)]</td><td>-VFORM
VFORM(SEG)=0</td><td>in(pv)=0</td></tr>
<tr><td>7.</td><td>in*</td><td>E*
E(SEG)=E(@1(SEG))</td><td>in(SEG)=in(@1(SEG))</td></tr>
<tr><td rowspan="6">attributes</td><td>8.</td><td>an [(pv)]</td><td>GOAL
GOAL(SEG).NE.0</td><td>an(pv).NE.0</td></tr>
<tr><td>9.</td><td>¬an [(pv)]</td><td>¬GOAL
GOAL(SEG).EQ.0</td><td>an(pv).EQ.0</td></tr>
<tr><td>10.</td><td>'nrn'</td><td>'BE'
@1(SEG).EQ.'BE'</td><td>@1(SEG).EQ.'nrn'</td></tr>
<tr><td>11.</td><td>¬'nrn'</td><td>¬'BE'
@1(SEG).NE.'BE'</td><td>@1(SEG).NE.'nrn'</td></tr>
<tr><td>12.</td><td>[pv_1] $ [an] pv_2</td><td>SIZE$'RELSIZ'
@1(SIZE(SEG))
.EQ.'RELSIZ' | etc.</td><td>an(pv_1).EQ.pv_2 |
an(an(pv_1)).EQ.pv_2 | etc.</td></tr>
<tr><td>13.</td><td>[pv_1] ¬$ [an] pv_2</td><td>SIZE¬$'RELSIZ'
@1(SIZE(SEG))
.NE.'RELSIZ',etc.</td><td>an(pv_1).NE.pv_2,
an(an(pv_1)).NE.pv_2,etc.</td></tr>
<tr><td rowspan="4">indicators</td><td>14.</td><td>in [(pv)]</td><td>PASSIVE
PASSIVE(SEG).NE.0</td><td>in(pv).NE.0</td></tr>
<tr><td>15.</td><td>¬in [(pv)]</td><td>¬PRM
PRM(SEG).EQ.0</td><td>in(pv).EQ.0</td></tr>
<tr><td>16.</td><td>in*</td><td>E*
E(@1(SEG)).NE.0</td><td>in(@1(SEG)).NE.0</td></tr>
<tr><td>17.</td><td>¬in*</td><td>¬E*
E(@1(SEG)).EQ.0</td><td>in(@1(SEG)).EQ.0</td></tr>
</table>

Figure 3.13. Notational Conventions in Specification Elements

designators in the pv parts. In most cases in the figure a pv part has brackets around it to indicate that it may be omitted, according to the convention already described. In some sense each of these 17 forms may be considered to be a type of "statement" available in this language. Three others would be the basic ones an(pv)=v, in(pv)=v, and v.r.v.

The first form shown in the figure is handy to use for "copying" the value of just one attribute from one record to another. It avoids having to write the attribute name twice. The second form is for "erasing" an attribute. Giving an attribute a value of zero is equivalent to erasing it because attributes with zero values are not stored in a record, i.e. the absence of a cell with a particular attribute number means that attribute has a value of zero. The element AGENT=AGENT-AGENT would also result in erasing the AGENT attribute, i.e. removing its cell from the record. Attribute 2 which holds all of the indicators is an exception to this. It is possible by turning indicators off to end up with a zero value cell for attribute 2.

The third form can be used to make attribute 1 point to some named record. Because of the important role that the SUP plays in the current application of this system, this form appears often in the listings in the appendices. It should be noted that the name SUP is <u>not</u> a special name to the system. Attribute 1 is given some special treatment by the system, but to call it SUP (or anything else) in the rules requires that the appropriate declaration be made, as in Figure 3.8 or Appendix A.

The fourth form is used to make the current segment record be a copy of some other record. (The percent sign should be read as "copy.") It usually appears as the first element in the specification, and usually copies either the segment record from the left of the rule (i.e. the one which just came off the stack) or some record pointed to from an attribute of that segment record (often an IPD record). Usually the new copy is then modified by successive creation elements. Clearly, copying could be accomplished one attribute at a time using form 1.

The next three forms have to do with setting indicators. The first one is for turning them on, the second is for turning them off, and the third is for getting their values from a named record. Turning on a multi-position indicator (NUMB, for example) means setting all its bits to ones; turning it off means setting them all to zeros. The binary constant shown with form 5 actually contains 55 one-bits, and as many as needed are taken from the right.

Form 7 takes into account the fact that an indicator name may have two different pairs of indicator numbers associated with it, one for segment records and one for named records, as discussed in Section A.7 of Chapter II. The "equivalent" form shown actually would not take that fact into account and would treat the record pointed to from attribute 1 as a segment record. This means that really those two forms are not completely equivalent for indicator names that are declared differently for segment records and named records in the manner described

in the previous chapter. None of the declarations listed in Appendix A are of this type.

It should be noted that forms 2, 5, and 6 can be used to affect records other than the current one, even though the examples given there do not demonstrate that. For example, -PTR(MEM) would erase the PTR attribute of the MEMORY record; -LC(PT('XY')) would erase the LC attribute of the record which is pointed to by the PT attribute of 'XY'; PASSIVE(PT('XY')) could set the PASSIVE indicator in that record.

It should also be noted that attribute 2 which holds all of the indicators can be treated just as any other attribute when convenient. For example, it may be copied or erased just as any other. The type of cell used by the system for an attribute 2 may be 0, 1, or 2 depending upon the highest numbered indicator which has ever been "on" in that record, as described in Section A.3 of Chapter II, but this should be of no concern to the user. The name INDIC is declared for attribute 2 in the listing in Appendix A and is used in some rules in the other appendices.

The remaining ten specification elements shown in Figure 3.13 are condition elements, i.e. they are special cases of v.r.v. It can be seen that they are in complementary pairs, with the second one of each pair having a "not" sign (¬). The first pair, forms 8 and 9, inquire about the existence or non-existence of an attribute. As was pointed out earlier, non-existence and zero-value are equivalent. Form 8 results in a "true" if the record has the designated attribute, and form 9

results in a "true" if it does not.

The next pair, forms 10 and 11, inquire about the value of attribute 1, with 'nrn' standing for 'named-record-name'. The special role of attribute 1, which is called the SUP in the current application, was mentioned earlier. Form 10 results in a "true" if attribute 1 of the current segment record points to the designated named record, and form 11 results in a "true" if it does not.

The pair of forms 12 and 13 inquire about set membership, i.e. is some particular record a member of some particular set. The record may be designated by any pv. If no pv appears before the $, SEG is assumed. The $ may be immediately followed by an attribute <u>number</u> to designate which attribute is used to point to a record representing the superset, i.e. the next record in the superset chain. If no number is given, a 1 is assumed. The last part of the element is a pv to designate the set. This is usually in the form of a named record name, but it need not be. For instance, it could be an attribute designator for an attribute which points to some named record.

Asking if a record is in some set usually means asking if "its SUP" or "the SUP of its SUP" or "the SUP of the SUP of its SUP", etc., points to the named record representing the set. The test must be carried out up through the chain until either the designated set record is found or the end of the chain is reached (i.e. a record with no SUP is encountered). The version with no pv before the $ is used quite heavily in the rules listed in the appendices (e.g. $'ACTION' or $'QUANVAL').

Form 12 results in a "true" if the record is in the set, and form 13 results in a "true" if it is not. These two forms are probably the most "powerful" in the language.

The last four forms given in the figure deal with the testing of indicators. The pair 14 and 15 simply ask whether an indicator is "on" or "off". An indicator is considered to be "on" if <u>any</u> of the bits in its range are <u>non-zero</u>; it is considered to be "off" if <u>all</u> of the bits in its range are <u>zero</u>. For example, with NUMB, SING, and PLUR declared as they are in Figure 3.8, NUMB would be considered to be "on" only if either or both of the others were "on". Form 14 results in a "true" if the designated indicator is "on", and form 15 results in a "true" if it is "off".

The last pair 16 and 17 are similar to 14 and 15 but are very specific with regard to where the indicators are located, i.e. in the named record pointed to from attribute 1 of the current segment record. These two forms are similar to form 7 in that they take into account the fact that an indicator name may have two different pairs of indicator numbers associated with it, one for segment records and one for named records. The discussion given there is applicable here too. Form 16 results in a "true" if the designated indicator in the named record is "on", and form 17 results in a "true" if it is "off".

Form 16 could have been used in the set of rules given in Figure 3.7. In this case the elements E* and EN* would not appear in the creation specification of rule 11. Also, the elements E and EN in the condition specifications of rules

18, 20, and 21 would be given as E* and EN* instead. With this approach those indicators would be tested in the relevant named record, rather than being copied into a segment record and then tested there.

It should be noted that forms 8, 9, 12, 13, 14, and 15 can perform tests in records other than the current one, even though most of the examples given in Figure 3.13 do not demonstrate that. For example, ¬PTR(MEM) would result in a "true" if there were no PTR attribute in the MEMORY record; ¬LC(PT('XY')) would result in a "true" if there were no LC attribute in the record which is pointed to by the PT attribute of 'XY'; PASSIVE(PT('XY')) could result in a "true" if the PASSIVE indicator were "on" in that record.

It should also be noted that four of the forms appear twice each in Figure 3.13, once as a creation element and once as a condition element. These are forms 1 and 8, 3 and 10, 5 and 14, and 7 and 16. When these forms appear in a creation specification, i.e. in parentheses on the right-hand side of a rule or in a named record definition, they are considered to be creation elements. When they appear in a condition specification, i.e. in parentheses on the left-hand side of a rule, they are considered to be condition elements. The other forms, including the three basic ones, have only one interpretation no matter where they appear.

The final convention to be discussed here is known as the "automatic copy". When a segment type on the right of the arrow has the same name as the segment type on the left of the

arrow, the segment record created begins as a copy of the segment record from the left (i.e. the one whose pointer just came off the stack). This means that

NP( ) --> VP( ) NP( )

is equivalent to

NP( ) --> VP( ) NP(%NP,)

In the case when the automatic copy is associated with the last segment type on the right, as in the above example, and there is no reason to keep the old record from the left, the old record itself becomes the new one, rather than there being a copy made of it. This saves some processing time.

4(f). Routines

There is one other kind of specification element in addition to the 20 described so far that has not been mentioned yet. It is called a routine, and is a device to make the complete capability of the computer available to the rule writer. In order to use this device the user must do two things. One, he must make a declaration to associate a routine number with a routine name, similarly to declaring attribute and indicator names. Two, he must write the routine in FORTRAN and insert it in the appropriate place in the CRSEG subroutine, which serves as the interpreter for creation and condition specifications. When a routine element is encountered during execution, control is passed to the appropriate user-written routine by means of a computed GO TO statement using the routine number.

Routines may be used both in creation specifications and in condition specifications.

Routine name declarations for the current application of this system can be seen in Appendix A. Two of these routines are used in encoding and will be described later in this chapter. The others are used in decoding and will be described in the next chapter. Each of these routines is fairly short, and together they consist of fewer than 50 FORTRAN statements. Also, they appear only a very few places in the rules.

5. Compilation of Rules

Encoding rules are usually punched on cards to be entered into the system. A rule may extend over several cards, with a card that is blank in at least the first 11 columns being considered a continuation card. Rule numbers, although included in the figures and appendices, are not entered. A set of encoding rules must begin with any one of the following heading cards:

```
SEMOLOGY FOR ENCODING:
LEXOLOGY FOR ENCODING:
MORPHOLOGY FOR ENCODING:
```

The system does not make any distinction among the three. Additional words (e.g. ENGLISH) may be included just prior to the colon.

Compiling a rule converts it into an internal format which is basically a string of bytes. The information in such a string is in essentially the same order as in the original rule, but is not in character form. For instance,

two-byte pointers to segment-type records appear in place of segment type names, and commands in a hypothetical machine language appear in place of the condition and creation specifications. (That is why these specifications have been referred to as being "procedures".)

Each command consists of a one-byte operation code and an operand field consisting of zero, one, or two bytes. A typical command is one which turns on an indicator, and another is one which tests for the existence of an attribute. Currently there are 52 different operation codes.

Storing the rules in this manner tends to use a minimum of storage, and also tends to make rule "execution" fairly efficient. For instance, rule 1 in Figure 3.7 consists of 71 characters (each of which is a byte in the EBCDIC code), but would result in only 35 bytes in its internal format. Two bytes of each rule are used to keep a pointer to the next rule that has the same segment type on the left.

6. Execution of Specification Elements

The execution of a creation element results in a change of information somewhere in the cell structure (except for certain cases that may occur occasionally, like erasing an attribute that already does not exist or turning off an indicator that is already "off"). The execution of a condition element results in a "true" or "false" outcome which then determines what to do next. A condition specification (i.e. the information in parentheses on the left of a rule) usually

contains only condition elements, but it may also contain creation elements. Similarly, most creation specifications (i.e. the information in parentheses on the right of a rule) contain only creation elements, but they may also contain condition elements. This latter case, which is referred to as "condition on the right," is very important because it often enables one rule to do the work of two or more, as will be explained shortly.

The procedures of both creation specifications and condition specifications are executed, i.e. interpreted, by the same FORTRAN routine CRSEG. It is called from two different places in the encoding routine; one of these is in the loop for testing conditions on the left to find a rule to apply, and the other is in the loop for creating segment records on the right after finding a rule to apply. When it is called for testing conditions, the calling routine needs it to return a "true" or a "false" to announce whether the conditions for applying that rule are met or not, but when it is called for creating a segment record, the boolean value returned can be ignored.

The CRSEG routine can be thought of as executing each element in the specification, one after another, until certain terminating situations occur. It should be noted, however, that the execution of each element may involve executing more than one command. In the following it is convenient to think of the execution of a creation element as always resulting in an outcome of "true", in addition to whatever effect it has on the

information in the cell structure. Then there are essentially two terminating situations: (1) If execution of an element results in an outcome of "false" and the element is <u>not</u> followed by a vertical bar in the rule, then return a "false" to the calling routine. (2) If execution of an element results in an outcome of "true" and the element is followed by a vertical bar or the closing right parenthesis in the rule, then return a "true" to the calling routine. If neither of these terminating situations prevail, then the next element is executed.

Part of what the above paragraph says is that in a condition specification commas can be read as "and" and vertical bars as "or", but that the usual hierarchy of logical operations does not apply. It also says that it is possible to terminate execution of a creation specification without executing all of the elements, if a condition element at some point results in an outcome of "false". This is the case of the "condition on the right" already mentioned.

6(a). <u>Condition on the Right</u>

The following is a simple example of the use of a condition on the right:

VP --> TA(N=M(VP),¬N,N=3)

This rule says that a VP yields a TA with an N attribute equal to the M attribute of the VP. However, if N turns out to be zero, i.e. if the VP has no M attribute, then let the N attribute of the TA equal 3, which may be considered to be a default value.

The way this works is that the element N=M(VP) would first be executed. If M(VP) has some value, not zero, then N of the record being created would take on that value. Then when the next element $\neg$N is executed, it would result in an outcome of "false" ($\neg$N is equivalent to N.EQ.0), and execution of elements would be terminated, leaving N with the value of M(VP). However, if M(VP) is zero (which means, as should be recalled from an earlier discussion, that a cell would not exist for it), then no cell would be stored for N in the record being created. Then when the next element $\neg$N is executed, it would result in an outcome of "true", and execution would continue. The next element would give N the value 3, and then execution would be terminated because of the closing right parenthesis.

The rule given in the example above would be equivalent to the following pair of rules:

```
VP(M)  -->  TA(N=M(VP))
VP     -->  TA(N=3)
```

Here, if the VP has an M attribute the first rule would apply; otherwise, the second one would. While this simple example serves to illustrate the principle, the real economy of using a condition on the right can better be seen in the rules listed in the appendices.

7. The Special Segment-Type OUTPUT

There is one special segment type name which can be used in an encoding rule - OUTPUT. According to the encoding

algorithm of Figure 3.9, when a segment of this type comes off the stack it is treated specially with certain output operations taking place according to the values of attributes 11-15 of the segment record.

Logically, the output stream can be considered to be one long string of characters. Physically, however, the encoding routine maintains a buffer which can hold one line image, the length of which can be set by the user. Whenever the buffer is full, a line is entered into the output file, normally at the terminal. Also, when the stack becomes empty, the last line in the buffer is put out.

Attributes 11 and 12 of an OUTPUT segment can be used to accomplish the functions of a carriage return and tab when needed. The value of attribute 11 is the number of carriage returns to execute, and the value of attribute 12 is the column position to be "tabbed" to. These are useful for any sort of formatted output, such as starting a new paragraph or producing statements in a programming language that has a rigid card format, such as GPSS.

Attribute 13 is intended for putting out long fixed character strings, such as "canned messages". The only kind of value it can have when it exists is a string value, and the characters of the string are entered into the output stream.

Attributes 14 and 15 are intended for putting out numbers - the first for integers and the second for decimals - and can only have number values. The value of attribute 15 is considered

to be in parts per thousand and is put out with an appropriate decimal point. For example, if attribute 15 equalled 12153, the output would be 12.153. Negative numbers can be handled in both cases. An OUTPUT segment with no attributes is effectively a null segment.

As an example of the use of OUTPUT, if the following appeared on the right of an encoding rule

```
OUTPUT(@11=1,@13="ABCDEFGHIJ",@14=567)
OUTPUT(@11=2,@12=5,@13="KLM  ",@15=3-27)
```

the following output would be produced

```
ABCDEFGHIJ567
      KLM  -.024
```

Other examples of the use of OUTPUT can be seen in the rule listings in the appendices.

8. Encoding Rules Viewed as a Computer Program

A set of encoding rules may be viewed as a computer program written in a special programming language that utilizes an unusual control mechanism. Each rule is essentially a statement of the form

IF (Condition Specification) THEN Right Part ELSE

A group of rules with the same segment type name on the left can be considered to form the body of a subroutine by that name, e.g. VERBPH. Each subroutine, then, consists of a series of IF statements which determine what the subroutine is to do under various conditions. In each case what it does

primarily is to "call" the subroutines listed in the right part, e.g. VERB and VERBPH.

The "unusual" control mechanism is that when a subroutine is called, rather than transferring control to it immediately, its "address" is simply put on a list of routines to be executed (i.e. the stack). Then, whenever a routine finishes execution, control is transferred to the routine on the top of the list.

Another somewhat unusual feature is that rather than passing a parameter list to a subroutine, an entire data area is passed (i.e. a segment record). The data area is set up at the time the subroutine is called, and its address is put on the list along with the address of the subroutine. Then, when the subroutine is executed the data area is made available to it. Such a data area can be considered to be "local" to a particular execution of the routine. All data that is accessible through the MEMORY record and the named records can be considered to be "global".

The "program" as described so far would not produce any output. In order to handle this, one thing that should be done is to insist that for each <u>non-terminal</u> segment type there be one rule that has no condition specification. If there is none already, a rule of the form

```
VERBS  -->  OUTPUT(@13=@10(@1(VERBS)))
```

should be added to take the place of part (ii) of the encoding algorithm. (Attribute 10 is the NAME attribute, as discussed in Section A.4 of Chapter II.) Also, for each <u>terminal</u> segment

type a rule of the form

```
A  -->  OUTPUT(@13="A")
```

should be added to take the place of part (a) of the algorithm. Then all output would be produced by the system subroutine OUTPUT, in the manner described in the previous subsection.

8(a). Comparison to Simulation Programming Languages

Although the control mechanism and the passing of data areas as described above may seem a bit unusual, they are actually quite similar to what is done for the typical simulation programming language. For example, in SIMSCRIPT the order in which subroutines (called "event routines") are executed is determined by a "calendar" of "event notices". Each event notice is a collection of attribute-value pairs, just as a segment record is, but stored differently in the computer. The calendar is a list of event notice identification numbers, similar to the stack, but ordered by event time, which is an attribute of an event notice.

Associated with each event notice is the address of an event routine, similar to the segment-type record pointer. (The actual implementation may be slightly different, but conceptually this is the way it is.) Whenever an event routine finishes execution, control is transferred to the routine on the top of the calendar, and the identification number of the associated event notice is made available to it. During execution of a routine, event notices may be created and given attribute values, and items (i.e. routine addresses and associated event notice

identification numbers for the same or different routines) may be inserted in the calendar by means of CAUSE statements, which may be thought of as "deferred" subroutine calls.

GPSS is similar, but the terminology is different. The order in which events occur is determined by the "future events list," which serves the same role as the calendar. The items on the future events list are "transactions", each of which is a collection of attribute-value pairs similar to an event notice. They are ordered by event time, also.

One of the attributes which a transaction has is the number of the next "block" that it is to go to. This serves the same purpose as a routine address, because in this language there are no event routines as such. Storing this number as an attribute of the transaction would be similar to storing the segment-type record pointer as an attribute of the associated segment record, and then having pointers just to segment records on the stack.

GPSS maintains another list of transactions, also, called the current events list, which consists of transactions that have come off the future events list but are waiting for something else. There are blocks in the language which when executed result in transactions being inserted in these lists.

The event notices of SIMSCRIPT and the transactions of GPSS may be thought of as local data areas, such as the segment records of this system. The method of referencing attributes of the local data area in this system has similarities to both of the others. When referencing an attribute of a transaction

in GPSS, the record need not be identified because the "current" transaction is the only one which can possibly be referenced. In this system when referencing an attribute of the current record, the record identification portion of the attribute or indicator designator (pv) may be omitted, according to the first convention described in subsection 4(e).

When referencing an attribute of the event notice which just came off the calendar in SIMSCRIPT, the name of the event routine is used to identify the record. This is similar to using the name of the segment type which is on the left of the rule to identify the record which just came off the stack when creating new records on the right of the rule, as discussed in subsection 4(c). The attribute designators of this system are similar in form to those of SIMSCRIPT, also.

Both SIMSCRIPT and GPSS have global data areas, too, but in general both of these languages use an array type of storage scheme for these areas, instead of the record scheme used for the local data areas. The distinction between these two schemes is basically that in the first one all values of <u>some attribute</u> for <u>all entities</u> are considered to be grouped together, whereas in the second one the values of <u>all attributes</u> for <u>some entity</u> are considered to be grouped together.

In the system being described here all data storage, both local and global, is basically of the record sort, but by means of the indirect specification of an attribute number, an array may be built within a record (e.g. the record lists in the IPD). This cannot be done in SIMSCRIPT. It can be done in GPSS, because

attributes of transactions are referenced by number (P1,P2, etc.) and "indirect addressing" is allowed. However, there is no way of referencing an attribute in any transaction other than the "current transaction," so this is of limited usefulness.

It is worth noting that attributes and indicators in this system are really just "variables", in the usual programming sense. An attribute or indicator designator furnishes a means of referring to a group of bits at some location in the computer, just as a variable name (possibly with a subscript) does in some other languages. At various times during program execution the same name may refer to different locations (e.g. SUP(SEG)), just as the dummy variables of a subroutine do, and different names may refer to the same location (e.g. SUP('XYZ') and @1('XYZ')), just as EQUIVALENCEd variables do. Attributes are similar to integer, character string, and pointer variables, and indicators are similar to bit string variables in some other languages.

8(b). Purpose of the Stack

Although the calendar (or future events list) is needed for controlling the order of execution of routines in a simulation programming system, the stack in its current form is not really needed for the system being described here. Whereas items are inserted into the calendar in a somewhat random order (according to the value of some time attribute), items are always only placed in a group on the top of the stack. This means that the order of execution of these

"routines" is completely known at the time they are placed on the stack. What it amounts to then is that the stack is really being used simply as a means of avoiding multi-level subroutine calls, including some which would be recursive.

In the earlier discussion it was stated that the right part in each "IF statement" consists primarily of calls to subroutines, but rather than executing each routine as it is called, its address and the address of its data area are put on the stack until the current routine is through. Clearly, another approach would be to execute each routine immediately. This would eliminate the stack as it is now, but would require keeping track of return points, which themselves may require a stack because of the possibility of recursive calls. It is not clear that there would be any advantage to using this other approach in the computer, but thinking about it does aid one's understanding of the system.

8(c). An Example of "Computing" with Encoding Rules

Figure 3.14 was prepared to furnish a simple example of the use of encoding rules to perform a rather conventional computation - the average of N numbers. At the top of the figure is a named record definition to furnish the data. The name of this record is completely arbitrary, of course, as long as its name and those of its attributes are consistent with the names used in the rules. For illustrative purposes, only three data points are given.

The encoding rules for this "program" appear next in the

Named Records:

```
DATA   (@11=15,@12=40,@13=20,N=3)
```

Semology for Encoding Average:

```
1   AVERAGE  -->  WORKREC(%'DATA',M=N+10,I=11)
2   WORKREC(I.LE.M)  -->  WORKREC(SUM=SUM+@I,I=I+1)
3   WORKREC  -->  OUTPUT(@11=1,@13="AVERAGE = ",
                      @14=SUM(WORKREC)/N(WORKREC))
```

Line	Rule	Output	Stack
1			AVERAGE(R1)
2	1		WORKREC(R2)
3	2		WORKREC(R3)
4	2		WORKREC(R4)
5	2		WORKREC(R5)
6	3		OUTPUT(R6)
7		AVERAGE = 25	

Records:		
	R1	(no attributes)
	R2	(@11:15,@12:40,@13:20,N:3,M:13,I:11)
	R3	(@11:15,@12:40,@13:20,N:3,M:13,I:12,SUM:15)
	R4	(@11:15,@12:40,@13:20,N:3,M:13,I:13,SUM:55)
	R5	(@11:15,@12:40,@13:20,N:3,M:13,I:14,SUM:75)
	R6	(@11:1,@13:"AVERAGE = ",@14:25)

Figure 3.14. Computing an Average Using Encoding Rules

figure. The first one serves to get it started, the second one provides the looping to compute the sum, and the third one computes and prints the average. Rule 1 expects there to be a record named 'DATA' in the form of the one shown above it, but with any number of points.

The lower half of the figure shows the execution of these rules using the 'DATA' record given, in the same form as Figures 3.10 and 3.11. In this case, the record R1 put on the stack initially along with the segment type AVERAGE would have no attributes other than a reference counter. The following command would begin the encoding algorithm by putting them on the stack:

ENCODE AVERAGE;

Then they would come off the stack and rule 1 would be applied, resulting in WORKREC and R2 being put on the stack. The record R2 would be a copy of the named record 'DATA', with attributes M and I added to it. Because it actually has no SUM attribute, it may be considered to have one with an initial value of zero, as discussed earlier.

When WORKREC and R2 come off the stack, rule 2 would apply because 11 is less than 13. This would result in a new WORKREC being put on the stack with R3. Actually, in the computer R3 would be physically the same record as R2, but with the SUM attribute added to it, as discussed at the end of subsection 4(e). With I equal to 11, the attribute designator @I is equivalent to @11.

The same rule would apply for two more iterations of the encoding algorithm (lines 4 and 5), and then rule 3 would be applied because 14 is <u>not</u> less than or equal to 13. This would result in OUTPUT and R6 being put on the stack. When they come off in line 7, output would be produced in the manner described in subsection 7.

If the following were then entered:

NAMED RECORDS:

DATA (@12=10,@14=5,@15=8,@16=7,N=6)

ENCODE AVERAGE:

Attributes 11 and 13 of 'DATA' would maintain their previous values, and the following output would be produced:

AVERAGE = 10

It can be seen that integer division truncates the result.

The above example is not meant to show that the encoding rule language should take the place of more conventional languages that are intended for doing these sorts of computations, but rather to show that it can do them when necessary as part of a more typical encoding task. What is especially important in the above example is the manner of doing looping. The same basic scheme is used quite often in the semological rules written for the queuing problem application, as will be seen in the next section of this chapter.

9. Comparison to Work by Others

Phrase structure rules, as discussed in subsection 1, are intended primarily for describing the strings of a language, rather than for generating them, and as such are rather static in nature. The encoding rules of this system, on the other hand, are intended for generating strings and therefore have a dynamic nature. In performing their task they have available a wide store of information and all of the procedural capabilities of the computer. Indeed, as was shown in the previous subsection, a set of encoding rules may even be viewed as being a computer program.

The main difficulty with using phrase structure rules to generate text is that there is no convenient facility for selecting a rule to apply out of those with the same symbol on the left. The introduction of "condition specifications" into these rules furnishes such a facility, and also takes care of the problem of infinite recursion. The introduction of "creation specifications" provides a facility for setting up the information to be tested in the condition specifications.

A very basic difference between the work done here and that discussed in subsection 1(b) is the order in which certain things are done. Both Yngve and Klein randomly establish the syntactic structure of a sentence first and then "plug in" some words. In Yngve's work the structure of the entire sentence could be determined and then words from the appropriate syntactic classes selected completely randomly later. In Klein's work, words were selected along the way and not completely

randomly, but still decisions about syntactic structure were made first. If the goal of the research is to explore the ramifications of some grammar, this may be a reasonable approach.

However, in the work being done here, the purpose of a set of encoding rules is to be able to take a specific piece of information in the form of an attribute-value semantic structure and convert it into equivalent information in the form of text in some language. In performing this conversion syntactic structures have to be established, but their form is determined by the information to be put out rather than the other way around. In certain situations where there is more than one form that could be used (e.g. active vs. passive), the decision as to which one to use could be made when writing the rules (e.g. by writing rules for only one form), or by including indicators in the initial semantic structure to select the desired one at the appropriate point in the processing (as was done for PASSIVE in the rules of Figure 3.7 and the example of Figure 3.11), or by systematically or randomly choosing different rules to apply from time to time in some particular situation (such as whether to put a subordinate clause before or after the main clause). In general it seems more reasonable to think in terms of concepts and relations first and then to choose an appropriate linguistic form, rather than vice versa.

Some other work which has been done within the same overall framework as this is the Relational Network Simulator of Reich [26]. His system is capable of converting a collection of

"signals", which correspond roughly to the indicators of this system, into a series of syllables. The reference cited above includes an example concerned with the English auxiliaries. Reich's system, which may be considered to be a Stratificational Grammar performance model, is intended strictly as a tool for theoretical investigations, however, and would not be able to handle language processing on the scale of the system being described here.

B. ENCODING RULES FOR THE QUEUING PROBLEM APPLICATION

In Section A of this chapter the encoding process was described in detail without regard to any particular application of this system. This included discussions of the encoding algorithm and the rule language. In this section six sets of rules which have been written for the queuing problem application are described. First there is the set for producing an English problem description, parts of which are also used by other sets. Then there is the set for producing a GPSS program, followed by the sets for "massaging" the IPD, asking questions, and answering questions. Finally, the small set which furnishes the linkage between decoding and encoding is described.

Listings of each of these six sets of rules appear individually in Appendices C through H. The rule numbers which appear in parentheses in the listings are not included as input to the computer but were placed there by a special listing program to make referencing easier. The declarations listed in Appendix A and the named record definitions listed in

Appendix B must be entered into the system prior to entering any of these rules. In order to make it easier for the reader to determine whether a name found in the rules is the name of a named record, an indicator, an attribute, or a routine, the names in each of these groups are listed alphabetically in Appendix J.

Output produced by these rules can be seen in the sample terminal session which appears in the Introduction. Reference to that output and to Figure 2.8, which shows the Internal Problem Description for the sample problem of Figure 1.1, can be helpful in following the discussions of this section.

1. Encoding the English Problem Description

The set of rules for encoding an English description of a queuing problem is listed in Appendix C. These rules are grouped into three strata - semology, lexology, and morphology. The semology basically segments the information in the Internal Problem Description into sentences in the form of segment records of type SENT, which it passes down to the lexology. The lexology basically converts a SENTence segment record into a series of word segment records, each of which is then converted by the morphology into a series of characters placed in the output stream. This lexology and morphology are similar to, but larger than, those given in Figure 3.7. The rules of each stratum are discussed in this subsection, followed by a discussion of some potential improvements to the English problem description.

1(a). <u>English Encoding Semology</u>

The English encoding semology does not directly produce any output. What it does do is create segment records with an associated segment type of SENT, which are passed to the lexology for expansion and eventual output. In general these records may have any of the attributes that action and entity records in the IPD have, plus some others like ATTRIB. The general scheme for making a statement about something in the IPD is to make a copy of the record of interest, add or delete some information, and call it a SENT.

There are basically two types of sentences considered - action sentences and attribute sentences. An action sentence is one which has an action verb, such as "unload", for its main verb, and furnishes information about several attributes of the action, such as the agent, goal, and location. For example, "The men unload the ship at the dock." An attribute sentence is one which has an attribute verb, such as "be", for its main verb, and furnishes information about one attribute, such as the duration of an action or the capacity of a stationary entity. For example, "The time for the men to unload the ship is eight hours."

In order to result in an attribute sentence, the SENT segment record must have an ATTRIB attribute to specify which attribute is of interest. The value of ATTRIB must be a pointer to the named record representing that attribute, usually one in the set 'ATTR'. For example, a SENT segment which is a copy of an action record and has an ATTRIB of 'DURATION' would result

in a sentence of the form, "The time for ... is ...." A SENT segment which is a copy of an entity record and has an ATTRIB of 'COLOR' would result in a sentence of the form, "The color of ... is ...." A SENT segment with no ATTRIB attribute results in an action sentence.

The first rule in the English semology produces six segment records. The first one, which is a copy of the stationary entity list, results in turning off the MARK1 and MARK2 indicators in each stationary entity record in the IPD, as will be discussed in detail shortly. The second segment record results in the same thing for mobile entities. Each of the next two records is a copy of the action list, the first for setting the MOBENATR attribute in each action record, and the second for actually producing the problem description. The last two segment records from the first rule have no attributes (other than the reference counter, attribute 0). The first of them simply results in indentation for a new paragraph, as can be seen in rule 186, and the other one results in the sentence which gives the problem time and basic time unit. The first four segment records are also given an LC attribute with a value of 11.

The MARK1 indicator is used to "mark" an IPD record during English encoding to signify that it has already been described or mentioned. The MARK2 indicator is currently not being utilized. These indicators are turned off initially in case they were left on from encoding a previous description. The clearing of these indicators will be explained in detail because

it furnishes a good example of the application of the looping technique described earlier for doing processing which is of an essentially non-linguistic nature. It is typical of what is done in many of the semologies being described here.

The first segment record of type CLEAR which comes off the stack is a copy of the stationary entity list. This was accomplished by the element %SEPTR(MEM) in rule 1. In Section C.1 of Chapter II it was pointed out that the value of SEPTR(MEM) is 'STALIST'. Actually, the element %'STALIST' would accomplish the same thing. This CLEAR segment, then, has attributes 11, 12, etc. pointing to stationary entity records, LASTREC giving the number of the highest attribute used, and LC which was set to 11 in rule 1.

When this first record comes off the stack, the condition in rule 3 would be evaluated, and if LC is less than or equal to LASTREC that rule would be applied. Because every queuing problem that can currently be handled by this system has at least one stationary entity, LASTREC will be at least 11 and this rule would be applied at this point. In applying the rule the new segment record to be built will begin as the one which just came off the stack. (This is the special form of the "automatic copy" as described previously.) Then each element in the creation specification will be executed.

Execution of the first element results in turning off the MARK1 indicator in some record. The pv part of the indicator designator is @LC, which is equivalent to @LC(SEG)(SEG). Because LC of this segment record equals 11 at this point, it is

also equivalent to @11(SEG), which is a pointer to the <u>first stationary entity record</u>. The second element turns off MARK2 in that same record, and the last element increments LC in the segment record to 12. Then that segment record is put back onto the stack as a CLEAR.

The next record to come off the stack is the CLEAR that was just put on. This still has the same attribute as the previous time it came off, except that now the value of LC is one greater than before. When the condition in rule 3 is evaluated, if LC is still less than or equal to LASTREC that rule would be applied again. This time however, because LC equals 12, the MARK1 and MARK2 indicators in the <u>second</u> stationary entity record would be turned off, and then LC would be incremented to 13. Again, the segment record would be put back onto the stack.

This looping would continue until the value of LC exceeded that of LASTREC, at which time rule 4 would apply. This would result in a segment of type NULL being put on the stack. Then the NULL would come off the stack, and rule 195 would apply, resulting in a segment of type OUTPUT with no attributes being put on the stack. When this OUTPUT segment comes off the stack, it would be given special treatment by the encoding algorithm as described earlier, but because it has no attributes 11-15, absolutely no output would result from it, i.e. the "null" string would be produced.

Rules 5-9 primarily give a value to the MOBENATR attribute of each action record in the IPD. MOBENATR is made to be either

'AGENT' or 'GOAL', depending on which one of those two attributes has a value in the set 'MOBENTY'.

Rule 10 "goes down the action list" to create a series of ACTN segments, each one of which is a copy of an action record from the IPD. The looping done here is basically the same as in rule 3, except that there are two segment types on the right side of the rule, resulting in two segments being put on the stack each time. The first one will be completely expanded, possibly resulting in several sentences being printed out, before the second one comes off the stack. Eventually, LC of the ACTLIST segment will exceed its LASTREC and the ACTLIST will "go to" NULL.

The first time rule 10 is applied LC(ACTLIST) equals 11, from rule 1. Therefore, the creation specification for ACTN

% @LC(ACTLIST)(ACTLIST)

is equivalent at that time to

% @11(ACTLIST)

which results in the ACTN segment record being a copy of the first action record in the IPD. The pair of rules 10 and 11 is typical of many in the other encoding semologies that have been written, as will be seen later.

Rules 12-18 decide in what manner the action is to be described, depending upon what sort of attributes it has. Each rule is basically of the same type discussed for VERBPH and NOUNPH in the example set of rules in Figure 3.7. Each one puts out something, followed by a new ACTN, with some

feature changed so that the same rule will not apply again.

If the action has not been mentioned before, rule 12 begins a new paragraph with a simple action sentence which includes the AGENT and/or GOAL and the LOCATION (e.g. "The vehicles arrive at the station."). If the IETM or DURATION attribute has a simple value, it will be included also, as a prepositional phrase (e.g. "every 8 minutes" or "for 5 minutes"). Otherwise, rule 13 or 15 will put out a separate attribute sentence with the value (e.g. "The time between arrivals ... is ...." or "The time for the vehicles to be serviced ... is ...."). If the stationary entity of the LOCATION has not been described yet, rule 14 produces a STENDESC, which is then processed by rules 32-34 to put out attribute sentences for the QUANTITY and/or CAPACITY in some cases (e.g. "There is 1 pump in the station." or "The capacity of the station is 10 vehicles.").

If the action has an ASNDISTR, rule 16 produces an ASNDESC which is a copy of the assignment distribution record. In rule 19 this becomes a RECLOOP1 preceded by a RECORD. It can be seen there that the RECORD is given a SUP of 'CMPLX', and then a pointer to the RECORD is stored in RP(MEM). It is important to realize that the word RECORD which appears in the creation specification is equivalent to SEGMENT or SEG, as described in subsection 4(c), and is not considered to be the segment type name RECORD which appears outside of the parentheses.

According to rule 42, RECORD goes to NULL, but that does not erase the segment record associated with it, because a pointer to that segment record was stored in RP(MEM). RECLOOP1 is processed by rules 35 and 36 to create records for each clause of the sentence which will eventually be put out (e.g. "75 percent of the vehicles are cars, and the rest are trucks."), and to place pointers to these records as the values of attributes 11, 12, etc. of the record pointed to from RP(MEM). Eventually, a RECLOOP is produced, and it yields a SENT which is a copy of the record that was just built (i.e. the one with a SUP of 'CMPLX' and with attributes 11, 12, etc. for the clauses), according to rule 41.

Finally, if the action has a SUCCessor, rule 17 produces a SUCCDESC which is a copy of it. The copy is given a PRED attribute pointing to the current action, to eventually result in a subordinate clause beginning with "after". A SUCCDESC is processed by rules 20-31 and 37-40, depending upon its type. The simplest case is when the SUCCessor is just another action, and rule 31 produces a SENT with the other action as its main clause (e.g. "After being serviced..., the vehicles leave the station.").

The other rules of that group handle the five types of successor descriptors which were discussed in Section C.6 of Chapter II. A 'PTYP' or a 'FRACTNL' results in a 'CMPLX' SENT similar to the ASNDESC described above, using RECLOOP2 and RECLOOP3. A 'QTYP', 'FTYP', or 'STYP' results in a pair of sentences, the first one with an "if" clause and the second one

beginning with "otherwise" (e.g. "After arriving at the station, if the length of the line is ..., the vehicle will be serviced .... Otherwise, the vehicle will leave the station.").

Rule 43 produces a SENT from the segment type ENDING which was produced by rule 1. This SENT is a copy of the MEMORY record, but with an ATTRIB of 'PROBTIME'. In the lexology this will get some special treatment to be expanded to a sentence of the form, "The simulation is to be run for ..., using a basic time unit of ...."

1(b). <u>English Encoding Lexology</u>

The task of the English encoding lexology is to take segment records of type SENT and expand them through several intermediate levels of processing to a series of segments of type VERB, NOUN, ADV, etc., which are passed to the morphology for further processing to result in the output of a stream of characters. SENT segments come not only from the English encoding semology just discussed but also from the semologies for asking and answering questions. They also come from the GPSS lexology, for putting comments into a GPSS program. These other sources are described in later sections.

It can be seen in the listing in Appendix C that the English encoding lexology begins with several rules (44-54) which handle the overall clause structure and punctuation of the sentence being produced. The first rule sets the QUESMK indicator for an interrogative sentence, which results

in a question mark at rule 53 instead of a period at rule 54. Rules 45-48 produce appropriate conjunctions and subordinate clause segments for sentences that have SUCCessor, PREDecessor, and CONDITN attributes, rules 49-51 produce the main clause or clauses, and rule 52 produces a "because" clause if there is a REASON attribute.

It was stated in the previous subsection that there are basically two types of sentences considered - action sentences and attribute sentences - and that an attribute sentence is produced from a SENT segment record with an ATTRIB attribute, where the value of ATTRIB is a pointer to the named recording representing the attribute of interest (e.g. 'DURATION'). In the lexology, rule 51 produces a FINCL (finite clause) segment which is a copy of a SENT, and then rule 55 checks for the existence of an ATTRIB in it. If there is an ATTRIB, the segment becomes an ATVCL (attribute-value clause) to be further processed; otherwise, it becomes a VERBPH immediately in rule 56. Rules 57-58 make a VERBPH with an appropriate indicator (PRESPART or INFIN) out of a PRPTCL or an INFINCL.

Rules 59-69 expand an attribute-value clause according to the type of attribute. If the ATVCL segment has an ATTRIB attribute but not the attribute of interest (e.g. if ATTRIB equals 'DURATION', but the record has no DURATION attribute), rule 59 produces an ATVQUES, which then becomes a FINCL with an appropriate interrogative phrase (INTRGPH) in rules 68-69. Certain attributes are given special treatment

in rules 60-65, and the others are handled by rules 66-67. In most cases a FINCL with a SUP of 'BE' is produced, which then would become a VERBPH by applying rule 56. A VERBPH with a SUP of 'BE' eventually results in a sentence whose main verb is a form of "be".

Rules 71-92 take care of putting out a VALUE. Depending on the type of value, this either results in an ADJ or NOUNPH directly or results in some intermediate segments which require additional processing. For instance, rules 74 and 82-89 participate in the production of phrases of the form, "normally distributed, with a mean of 10 minutes and a standard deviation of 2 minutes."

Rules 93-128 expand segments of type VERBPH. It should be noted that in these rules a VERBPH segment initially covers a whole clause, and that any subject, object, and modifying phrases will directly or indirectly come from it. If the VERBPH does not have a SUBJECT attribute, rules 94-96 will make the SUBJECT be either the AGENT or GOAL or simply "something". If the VERBPH does not have a value for its NUMBer, rules 97-99 will set either the SING or PLUR indicator, according to the characteristics of the subject. A VERBPH becomes a VERBPH2 in rules 101-103. Then in rule 104 two copies are made of the VERBPH2 to produce a VERBPH3 and a VERBPH4.

A VERBPH3 is processed in rules 105-115 to put out everything up to and including the main verb of the clause. This group of rules is basically like rules 4-7 in the example set

of rules in Figure 3.7, but handles more complicated verb phrases, including interrogatives and negatives. Rule 106 ensures that some form of "do" will be included in a phrase that needs one. Rule 107 inserts a "not" at an appropriate point when needed. Rules 108-111 put out auxiliary verbs that are called for. The subject is put out at the appropriate time by rule 112, and special cases of "be" are handled by rules 113-114. The main verb itself is produced by rule 115. It is important to realize that in general rules are not applied in the order in which they appear, but rather according to the conditions existing in the segment being processed. For example, both "cars are serviced" and "are cars serviced" would be put out by applying the same rules but in a different order.

A VERBPH4 is processed in rules 116-128 to put out everything after the main verb. The general form of these rules is to ask if the segment has a certain attribute and then put out a phrase (usually a prepositional phrase) with its value, deleting the attribute from the VERBPH4. For example, if the segment has an AGENT attribute, rule 117 puts out an agentive-by prepositional phrase. Rule 129 splits a PREPPH phrase into its PREPosition and NOUNPH object.

Rules 130-146 process a NOUNPH. The first three handle the special cases for percent (e.g. "30 percent of the ships"). Rule 133 treats quantitative values, such as "10 minutes". A condition-on-the-right, as discussed in Section A.6(a) of this chapter, is used to set the number of the noun properly. The next three rules handle "the" and "a", in a fashion

which is rather crude, but adequate for now. Rules 137-139 process a NOUNPH which has an ATTRIB attribute, to eventually produce a phrase of the sort, "color of the ship". The other NOUNPH rules essentially handle modifiers.

The final three rules in the lexology (147-149) give an appropriate SUP to certain modal verbs. For instance, a VERB with a FUTURE indicator gets a SUP of 'WILL'.

1(c). English Encoding Morphology

The task of the English encoding morphology is to convert word segments into character strings. The first seven rules (150-156) insert a space at the beginning of a word. Then there are several rules (157-165) which treat certain irregular verbs and add appropriate suffixes to regular verbs. Rules 166-168 perform a similar function for nouns. Most of these were included in the example set of rules in Figure 3.7.

The bulk of the remaining rules treat special cases, such as putting out certain words that are longer than eight characters. Also there are rules which yield segments of type OUTPUT, the special segment-type discussed in Section A.7, for segments of such types as NEWPAR (new paragraph), WORD, NAME, and NUMBER.

1(d). Improving the Style of the English Problem Description

The English problem description produced by this set of encoding rules has a rather rigid style, as can be seen in Figures 1.2 and 1.5 in Chapter I. It is adequate for its purpose of informing the user about what is in the IPD, and in

fact, its explicitness may even avoid some problems of misunderstanding on the part of the user, but it sounds like it was produced by a computer!

There are clearly a number of possibilities for making its style less rigid, however. For one, the order of the clauses could be varied, either randomly (by drawing and testing random numbers in the condition parts of some rules) or systematically (e.g. by keeping counters in the MEMORY record and testing their values in the condition parts of some rules). The rules that would be affected are at the top of the lexology (45-51).

Another possibility would be to use some pronouns in the encoded text. A simple scheme for doing some of this would be to always keep track of the mobile entity most recently mentioned by name in the encoded text (using an attribute of MEMORY), and each time a mobile entity is to be mentioned (i.e. a NOUNPH in the set 'MOBENTY' - without an ATTRIB attribute - comes off the stack) check to see if it is the one most recently mentioned. If it is, put out a pronoun instead of a noun phrase. Of course, the pronoun would have to be chosen carefully with regard to gender and case. Stationary entities could be dealt with similarly. In the English decoding rules, essentially the same scheme, but in reverse, is currently used to handle pronouns which appear in the input text, as is discussed in Section B of Chapter IV.

Another possibility for improving the style of the encoded English text would be to eliminate some "over-specification". For example, if there are cars in the model and all the cars are blue, then refer to them as "blue cars" only the first time they are mentioned; after that refer to them simply as "cars". For another example, if there are pumps in the model and they are all at the same place (e.g. in the station), then after referring to them as "pumps in the station" the first time, simply refer to them as "pumps" throughout the remainder of the description. The same sort of thing could be done with references to actions. A scheme somewhat similar to that described above for pronouns could be used here too because they both attack basically the same problem, that of "anaphoric reference".

It would be a fairly simple matter to write a few more encoding rules to implement these schemes, and they would probably work well in many or most cases. In fact, this is the philosophy that has been applied at a number of points along the way in order to get this system running. However, now that the overall framework has been established it would seem more appropriate to select particular topics (e.g. anaphoric reference) one at a time and do a detailed analysis of each one in order to arrive at solutions which could be more generally applicable. Such an approach would not only result in more substantial improvements being made to the simulation programming system, but could also result in significant contributions being made to linguistic theory.

2. Encoding the GPSS Program

The set of rules for encoding a GPSS program is listed in Appendix D. These rules are grouped into the same three strata as the English rules - semology, lexology, and morphology. The semology basically segments the information in the Internal Problem Description into GPSS statements in the form of segment records of type STMNT, which it passes down to the lexology. The lexology basically splits these segments into separate segments for each of the fields of a GPSS statement (i.e. label, operation, modifier, and arguments A-H). Then the morphology converts each of these into an appropriate series of characters in the output stream. The rules of each strata are discussed in this subsection.

2(a). GPSS Encoding Semology

The overall structure of the GPSS program produced by this set of rules can be seen in rule 201. After some initialization with INITIALGPSS and SETAGL, a SIMULATE statement and an RMULT statement are produced. Then there is an SELIST which is a copy of the stationary entity list of the IPD. This SELIST record results in making a pass down the stationary entity list in rules 204-205 to produce an EQU statement and a STORAGE statement in rules 208 and 210 for each stationary entity in the IPD. The MELIST similarly produces an EQU statement and a TABLE statement for each mobile entity, using rules 206-207 and 209-213.

Then there is an EXPFUNC and a NORMFUNC which result in FUNCTION definitions for the standard exponential and normal distributions when they are needed, using rules 214-217 and 222. The coordinates for these distribution functions are held in the named records 'EXPONREC' and 'NORMREC' which can be seen on the last page of Appendix B.

Then there is a DISTLIST which results in a pass down the distribution list in rules 218-219 to produce FUNCTION definitions. Similarly, the SUCLIST results in a pass down the successor descriptor list in rules 220-221 to produce some FUNCTION definitions. Rules 222-223 decide whether or not a FUNCTION definition is actually needed, and if so, produce a FUNCTION statement and a "function follower" statement.

Then there is a DSTLIST which results in another pass down the distribution list to produce an FVARIABLE statement for each normal distribution, using rules 224-227. This is followed by an ACLIST which results in production of the block statements by making a pass down the action list in rules 228-229. Finally there is a TMLOOP which produces the timing loop in rule 240.

The pass down the action list in rules 228-229 produces a series of ACTT segments, each of which is a copy of an action record. Then in rule 230, an ACTT produces a "comment" statement and an ACT. The latter of these is processed by rules 231-236 to produce a series of STMNT segments appropriate for that kind of action, followed by a SUCSTMNT, which

itself is processed by rules 237-239 to produce one or two STMNT segments according to the type of SUCCessor the action has. For instance, an activity which takes place at a location which has a QUANTITY and CAPACITY of 1 would result in QUEUE, SEIZE, DEPART, ADVANCE, and RELEASE statements, using rule 233; a 'QTYP' successor descriptor would result in TEST and TRANSFER statements using rule 237.

The remaining rules in this semology (241-246) modify or delete certain STMNT segments, with the last rule essentially just changing the name to GSTMNT. For instance, rule 242 deletes a TRANSFER statement that would simply be transferring to the next statement in the GPSS program, and rule 243 establishes the appropriate values for the A and B arguments in the case of a uniform distribution.

2(b). GPSS Encoding Lexology

Primarily, the GPSS encoding lexology converts a GSTMNT into a series of smaller segments, one for each field of the statement, and takes care of format. However, it also gives special treatment to comments and function followers. Most GSTMNT segments go through rules 253-255 which handle the splitting and formatting in a straightforward manner. The LABFLD, MODFLD, and ARGFLD segments produced are then processed by rules 256-261 to produce ARG segments when needed.

Comments are handled by rules 250-252. In rule 252 a SENT segment is produced to be further processed by the English encoding lexology, as already discussed.

Function followers are handled by rules 247 and 262-265. Each time through rule 263 an x-y pair is put out as two ARG segments with proper punctuation. In this rule the standard exponential and normal distributions are given slightly different treatment than other functions are.

2(c). <u>GPSS Encoding Morphology</u>

Most of the rules in the GPSS encoding morphology have to do with processing an ARG segment passed down from the lexology. In rule 268 a Routine named GETYPE is called to determine the type of value which the DATA attribute of the ARG segment has. This FORTRAN Routine simply gets the value of the TYPE field of the cell for attribute 9 of the current segment and stores it as the new value of attribute 9, which can then be tested in the usual fashion. If the DATA attribute is a pointer, then a PARG which is a copy of the record pointed to is produced in rule 268. Rules 269-270 handle the cases of numeric and string values of the DATA attribute.

The various kinds of PARG segments are processed in rules 271-284. Most of these produce character strings, either directly or by using segments such as NAME, NUMBER, and DECNUMB which result in appropriate OUTPUT segments by applying rules 190-195 in the English encoding morphology. A PARG in the set 'ABSTIME' (e.g. "10 minutes") yields a TIMARG segment in rule 274. This is processed by rules 285-290 to put out the proper number taking into account the basic time unit specified for the model.

The other rules in this morphology handle the other types of segments passed down from the lexology. Rules 266-267 take care of two OPFLD segments whose names are more than eight characters long. Rules 291-292 insert the correct number of commas between arguments, taking into account null arguments. Finally, rules 293-299 take care of card format, utilizing the special segment-type OUTPUT.

3. Massaging the IPD

The set of rules for "massaging" the IPD is listed in Appendix E. These rules are all considered to be semological. They do not produce any output either directly or indirectly through some lexology and morphology. Their purpose is to "clean up" the Internal Problem Description to prepare it for the production of a GPSS program or an English problem description using the sets of rules already discussed. The "massager" accomplishes its task by going down each of the seven lists in the IPD to examine each record and possibly modify it.

Rule 301 produces an INITIALIZE segment and seven RECLIST segments, each of which is a copy of a list record from the IPD. In rule 302 an INITIALIZE segment results in the initialization of several indicators and attributes in the MEMORY record. From the RECLIST segments rules 303-304 make copies of the individual IPD records, one at a time, calling each a REC. Then in rules 305-313 these REC segments become segments of unique types corresponding to the

type of record (e.g. ACTNREC). Each of these segment records has an ENTY attribute which points to the IPD record which this segment record is a copy of. It is through this ENTY attribute that changes are made to the IPD records themselves in later rules.

Rules 314-320 process ACTNREC segments. For instance, rule 314 adds an ASNDISTR attribute to an 'ARRIV' action when it is appropriate; rules 316-317 change a TIME attribute to either a DURATION or IETM attribute, depending on whether the action is an activity or an event. Rules 321-330 process MOBREC and STAREC segments, primarily filling in default values of certain attributes such as CONSUMPtion, QUANTITY, and CAPACITY. Rules 331-336 primarily assign values to the IDNO attributes of those distribution records and successor descriptor records that will result in FUNCTION or FVARIABLE statements in the GPSS program.

Rules 340-342 give a value to the IDNAME attribute of a mobile or stationary entity record. This makes use of the FORMSYMBOL Routine, which concatenates the first three or four characters from the string in attribute 8 with the number in attribute 9 to form a new string which is then stored in attribute 8 (e.g. "TRUC3"). Rule 344 effectively erases a record which is pointed to only from a list record, and which therefore is not needed, by replacing the pointer with one to the named record 'NULL'.

4. Asking Questions

The set of rules for having the system ask questions is listed in Appendix F. These rules also are all semological and have a flow of control which is basically similar to the massaging rules just discussed. However, these usually result in output being produced through the English encoding rules discussed earlier. The task of this set of rules is to examine each record in the IPD to see if it has missing or erroneous information, and if so, to ask a question to rectify the situation. All of the questions asked by the system in Figures 1.1 and 1.4 (e.g. 8, 33, and 45) were initiated by these rules. The user responses were processed by the decoding rules which are discussed in Section B of Chapter IV.

Rules 401-408 serve a purpose similar to rules 301-313 in the previous set. It can be seen that they result in making two passes down the action list, followed by one each down the mobile and stationary entity lists. Also, a MEMREC1 is produced. The mobile and stationary entity list passes do not really accomplish anything, but were included for future expansion.

Rules 409-422 examine an action record during the first pass down the action list. Some of these rules check for missing information and others check for erroneous information either directly or by producing another segment which must pass through other rules. For instance, rule

411 asks if the action record has a LOCATION attribute, and if it does not, an INTSENT1 segment which is a copy of the action record but with an ATTRIB attribute of 'LOCATION' is produced. Also, the SUPSET attribute of MEMORY is given the value 'LOCDESCR', to be used by the English decoding rules to accept a phrasal reply only if it is in the set 'LOCDESCR' (e.g. "at the station").

The INTSENT1 segment produced by rule 411 would then be processed by rule 476 to yield a SENT segment, which in turn would be processed by the English encoding lexology and morphology to yield a question of the form "Where are the cars serviced?" The indicator QUESTSW(MEM) is set in rule 476 to prevent any further questions from being asked before this one is answered. This indicator is used in rules 402, 421 and others (including 1 and 201) to result in having all segments which remain on the encoding stack go to NULL at this time. Then decoding of the user's response can begin. Some discussion of the interplay between decoding and encoding is given later in Subsection 6 of this section, and more is included in the next chapter.

Also in rule 476 three attributes of MEMORY (LASTME, LASTSE, and LASTLD) are given values to be used in identifying the referents of pronouns when decoding the response. The use of these is discussed in Section B of Chapter IV. Also two other attributes of MEMORY (CENTY and ATTRIB) are given values to identify the record and attribute whose value is being sought. These can be used by the decoding rules in processing the response too.

A rule such as 413 applies for checking an attribute's value. In this particular case a QUANREC is produced, basically as a copy of the value record but with some additional information that will be needed for asking a question if an error is found. It also has a VALSET attribute which in this case specifies that the value of the attribute being checked (i.e. IETM) must be in the set 'ABSTIME' (e.g. "10 minutes"). In rules 433-437 QUANREC segments are processed. For instance, if the QUANREC is a 'STDIST' (e.g. 'NORMAL'), it becomes a DSTRREC1 in rule 434. Rules 438-447 process these segments to make sure that a distribution record has all of its parameters (e.g. MEAN), and that their values are in the set specified by VALSET.

In rule 419 a SCSRREC1 is made from the SUCC attribute of an action. Then in rules 448-459 a segment of this type is processed to make sure that a successor descriptor record has no missing or erroneous attributes. As part of this processing, each action record that is reached as a successor is marked by turning on its REACHED indicator. Then during the second pass down the action list, rules 423-424 issue a question about an action other than an 'ARRIV' or 'ENTER' which does not have this indicator on. Such a question would be of the form, "Before being serviced at the pump, what do the cars do?"

Rules 460-474 are involved in the checking of distribution records and successor descriptor records that will result in a FUNCTION in the GPSS program. They check for such

things as missing points and cumulative distributions which do not end in 1.0.

Most of the error statements and questions go out through rules 475-478. Each of these produces a SENT segment which is then passed to the English encoding lexology. As discussed above, a number of attributes of MEMORY are set up at this time to be used by the decoding rules in processing the response.

5. Answering Questions

The set of rules for having the system answer questions is listed in Appendix G. This set of semological encoding rules is different from the four sets already described, in that it does not involve going down any lists in the IPD. These rules are applied after a user-asked question is decoded. They determine what would be an appropriate response to the question and then initiate the production of this response, with the bulk of the response being produced by the English encoding rules discussed earlier. All of the answers given by the system in Figure 1.4 (e.g. 55, 58, and 61) were initiated by this small set of rules.

These rules do not do any searching of the IPD for information to answer a question. However, as is discussed in Section B of Chapter IV, a fair amount of searching is done during the normal decoding of a sentence, to relate phrases in the sentence to records in the IPD. The QUESTION segment with which this set of rules begins is a copy of a SENT

segment produced by the decoding rules, and already has pointers to all of the information needed for answering certain kinds of questions directly. These do not involve making inferences. The thrust of this research was not in the direction of question-answering, but this small amount was included to illustrate the system's potential in this area.

The QUESTION segment with which these rules begin would have an attribute 11 pointing to a segment record developed from the main clause of the question asked. Also, it might have a PRED attribute or a SUCC attribute pointing to a segment record developed from a subordinate clause beginning with "after" or "before". The kind of attributes in the segment developed from the main clause would depend on whether it was an action sentence or an attribute sentence. If it were an action sentence, it would have attributes typical of an action record, i.e. a SUP in the set 'ACTION' and other attributes like AGENT, GOAL, LOCATION, IETM, and DURATION. It would also have an ENTY attribute pointing to an action record in the IPD if there is one with the same attributes.

If it were an attribute sentence, the segment record developed from the main clause would have basically three attributes - ENTY, ATTRIB, and VAL. ENTY would point to the referenced record in the IPD, ATTRIB would point to the named record representing the attribute of interest, and VAL would point to the segment record developed from the value phrase in the sentence. This last segment record would also have an

ENTY attribute pointing to the corresponding value record in the IPD. The development of these records is discussed in detail in Section B of Chapter IV.

Rules 501-502 ask whether or not an IPD record about which the question is being asked has been identified. If it has, a QUEST1 is produced in rule 502. This segment is made as a copy of the IPD record, and then ATTRIB and VAL are added to it from the segment record associated with the main clause, if they exist. Rules 503-509 "massage" an action QUEST1 segment to give it a MOBENATR attribute and to make sure that a TIME attribute is interpreted correctly as either IETM or DURATION, depending upon whether an event or an activity is involved. Then in rule 510 a QUEST1 becomes a QUEST2.

Rules 511-517 handle the various cases according to the information available in the QUEST2 segment to produce an appropriate response. Rule 511 answers "Yes, ..." to a yes-no question that does not involve a particular attribute. Rule 512 answers "I don't know." when the record in the IPD does not have an attribute being asked about. Rule 513 produces a statement with the value of an attribute being asked about. Rule 514 answers "No, ..." to a yes-no question about a particular attribute. Rules 515 and 516 are similar to rules 512 and 514, but treat the QUANTITY attribute specially. Finally, if none of the other rules could be applied, rule 517 answers "Yes,...." to a yes-no question about a particular attribute.

Rules 518-523 answer questions of the sort "After arriving, what do vehicles do?" The segment-type SUCCDESC produced in rule 522 is processed by rules 20-25 in the English encoding semology. Rules 524-528 determine whether the sentence to be put out will be an action sentence or an attribute sentence, depending on the attribute of interest and its value. Rules 529-532 handle the "canned message" portion of the response.

6. <u>The Linkage Between Decoding and Encoding</u>

The small set of rules which furnish the linkage between decoding and encoding is listed in Appendix H. As will be seen in the discussion in Chapter IV, it is possible to initiate encoding from the decoding rules by having the application of a decoding rule yield a segment of the special type ENCODING. When this occurs, the encoding routine is called with the segment-type ENCODING and an associated segment record pointer on the stack. Then encoding rules with ENCODING on the left, such as 551-557, are used to begin the desired processing.

As is discussed in Section B of Chapter IV, in the queuing problem application the decoding of certain kinds of sentences results in a segment of type ENCODING. In some cases this segment is given an ETYPE attribute, and in some other cases an ETYPE attribute is stored in MEMORY. As can be seen in rules 551-557, the value of this attribute determines what type of encoding is to be done. This set of rules could be considered to be at a slightly higher level

than the five sets already discussed, because the top segment-type in each of those (i.e. the segment-type which appears only on the left side of rules, such as ENGLISH or QUESTION), appears on the right side of rules in this set.

Rule 554 is typical of the rules in this set. If a sentence requesting a GPSS program were decoded, ETYPE(MEM) would be given the value 'GPSSPROG', and a segment of type ENCODING would be produced by the decoding rules. Then the encoding routine would be called, and rule 554 would be applied, putting the three segment-types MASSAGER, INTERROGATOR, and GPSSPROG on the stack. Then the first of these would come off the stack, and rule 301 in Appendix E would be applied, resulting in the IPD massaging discussed earlier. After the massaging, INTERROGATOR would come off the stack, and rule 401 in Appendix F would be applied, resulting in the IPD being inspected for missing or erroneous information.

If no error were found by the "interrogator" rules, when GPSSPROG comes off the stack, rule 201 in Appendix D would be applied, resulting in the GPSS program being encoded. If an error were found, however, a question would be encoded and QUESTSW(MEM) would be set by the interrogator rules. Then when GPSSPROG comes off the stack, rule 202 would be applied instead of 201, yielding a NULL. After that segment is processed by rule 195, the encoding stack would be empty and a return would be made to the decoding routine to process the user's next statement or reply.

C. SUMMARY AND DISCUSSION

The first section of this chapter described the encoding process without regard to any particular application of this system. This included discussions of the encoding algorithm and the rule language. It was seen that encoding rules are basically phrase structure rules, but with a significant difference. Rather than simply names, the objects referred to in the rules are arbitrarily complex structures of attribute-value pairs which can be built and interrogated from the rules. The purpose of a set of encoding rules is to be able to take a specific piece of information in the form of an attribute-value semantic structure and convert it into equivalent information in the form of text in some language.

The second section of this chapter described six sets of encoding rules which have been written for the queuing problem application. These rules are for producing an English problem description, producing a GPSS program, massaging the IPD, asking questions, answering questions, and furnishing linkage between decoding and encoding. The English lexological and morphological rules may be considered to be the most important of these because they are also used by the other sets.

The explicit stratification in the rules for the queuing problem application is considered to be significant. A benefit derived from this stratification is that the English encoding lexology and morphology given here could be used for some other application. It is likely that they would have

to be modified somewhat because certain parts of them are peculiar to the terminology of queuing problems. However, some parts, such as the treatment of overall sentence structure and especially that of verb phrases, could probably be used just as is.

There are a number of improvements that could be made to the encoding rules for the queuing problem application. The "massager" could be eliminated as a separate entity, with most of its functions being incorporated into the beginning of the GPSS semology. Some other parts of what it does could be done from the English <u>de</u>coding rules. The "interrogator" could be improved by keeping track of where it is looking in the IPD at any time, rather than starting at the beginning after each response.

The order of both the English problem description and the block portion of the GPSS program are determined by the order in which the actions appear on the action list in the IPD. In both cases a more natural flow could probably be obtained by following "successor paths" instead. Other suggestions for improving the style of the English problem description were given in Subsection B.1(d).

The encoding rules given here are adequate for the kinds of queuing problems which this system can currently handle, i.e. those problems which can be described by the IPD records discussed in Section C of Chapter II. Expansion of the

system's capabilities would require modifications and additions at various points throughout these sets of rules, but it is unlikely that their overall format would have to be changed.

IV. DECODING

In this system decoding is the process of converting information in the form of text into equivalent information in the form of records in the cell structure. Sets of decoding rules written in the system's rule language are used to specify the exact manner in which decoding is to be done for any particular application. These rules must first be compiled into their internal computer format by the same FORTRAN routine that compiles encoding rules, and then they are used by the decoding algorithm, which is another FORTRAN routine, to process input.

This chapter is organized similarly to the previous chapter. It begins with a description of the decoding process without regard to any particular application of the system. Then there is a section describing the set of English decoding rules developed for the queuing problem application. Even though the decoding and encoding algorithms are dissimilar, the decoding rule language is essentially the same as the encoding rule language which was described in detail in Section A of the previous chapter. This chapter assumes that the reader is familiar with that material, and therefore *cannot* be read independently of it.

A. THE DECODING PROCESS

Decoding is the inverse process of encoding. Whereas the *encoding* algorithm uses encoding rules to *split* higher level segments into contiguous lower level segments, the *decoding* algorithm uses decoding rules to *combine* contiguous lower level

segments into higher level segments. In this section the decoding rule language and the decoding algorithm are described using a small set of English rules and examples of their application for illustration.

1. Decoding Rules

Figure 4.1 (2 pages) shows a set of decoding rules for roughly the same portion of English as the encoding rules of Figure 3.7 and the phrase structure rules of Figure 3.1. The similarities and differences among these three sets of rules may be readily noted, with probably the most noticeable difference being that in the decoding rules the constituents appear on the left rather than on the right. Also, this set of rules has a semology in addition to a morphology and lexology.

Figure 4.2 contains the relevant declarations for the rules of Figure 4.1 and is identical to Figure 3.8. It may be noted that some of the names which appear in these rules (e.g. SUBJECT, OBJECT, and LC) do not appear in the declarations. This is just a case of implicit symbolic attribute specification as discussed in Section A.4(b) of Chapter III.

A decoding rule specifies what to do when a particular series of contiguous segments appears in the input text. Each segment is referred to by segment type name, and a condition specification may be given in parentheses with each one. The "what to do" basically consists of giving a name (the segment type on the right) to a new segment which covers (i.e. includes) the segments referred to on the left of the rule and creating a record to describe the new segment according to the creation

<u>Morphology for decoding English:</u>

```
 1   B  O  Y  -->  NOUNS('BOY')
 2   B  O  A  T  -->  NOUNS('BOAT')
 3   S  H  I  P  -->  NOUNS('SHIP')
 4   A  R  R  I  V  -->  VERBS('ARRIV',SUFX*)
 5   S  E  R  V  I  C  -->  VERBS('SERVIC',SUFX*)
 6   U  N  L  O  A  D  -->  VERBS('UNLOAD',SUFX*)
 7   L  O  A  D  -->  VERBS('LOAD',SUFX*)
 8   H  A  V  -->  VERBS('HAV',SUFX*)
 9   B  E  -->  VERBS('BE',SUFX*)
10   B  I  G  -->  ADJP('BIG')
11   T  H  E  -->  ADJP('THE')
12   NOUNS  -->  NOUNP(%NOUNS,SING)
13   NOUNS(¬ES*)  S  -->  NOUNP(%NOUNS,PLUR)
14   M  E  N  -->  NOUNP('MAN',PLUR)
15   M  A  N  -->  NOUNP('MAN',SING)
16   VERBS(NSFX)  -->  VERBP(SUP(VERBS),PLUR)
17   VERBS(S)  S  -->  VERBP(SUP(VERBS),SING)
18   VERBS(E)  E  -->  VERBP(SUP(VERBS),PLUR)
19   VERBS(ES)  E  S  -->  VERBP(SUP(VERBS),SING)
20   VERBS(ED)  E  D  -->  VERBP(SUP(VERBS),PASTPART)
21   VERBS(EN)  E  N  -->  VERBP(SUP(VERBS),PASTPART)
22   VERBS(ING)  I  N  G  -->  VERBP(SUP(VERBS),PRESPART)
23   H  A  S  -->  VERBP('HAV',SING)
24   A  R  E  -->  VERBP('BE',PLUR)
25   I  S  -->  VERBP('BE',SING)
26   .  -->  PUNC
27   #  -->  PUNC
```

Figure 4.1. A Set of English Decoding Rules
(cont'd on next page)

Lexology for decoding English:

```
28   # ADJP  /PUNC  -->  ADJ(%ADJP,ATTRIB($(SEG)))
29   # NOUNP  /PUNC  -->  NOUN(%NOUNP)
30   # VERBP  /PUNC  -->  VERB(%VERBP)
31   NOUN  -->  NOUNPH(%NOUN)
32   ADJ(ATTRIB)  NOUNPH(¬@ATTRIB(ADJ),¬DETERM)  -->
                               NOUNPH(@ATTRIB(ADJ)=SUP(ADJ))
33   VERB  -->  VERBPH(%VERB)
34   VERB('BE')  VERBPH(PASTPART,¬PASSIVE,¬PROG,¬PERFECT)  -->
                          VERBPH(PRM,PASSIVE,VFORM=VFORM(VERB))
35   VERB('BE')  VERBPH(PRESPART,¬PROG,¬PERFECT)  -->
                          VERBPH(PRM,PROG,VFORM=VFORM(VERB))
36   VERB('HAV')  VERBPH(PASTPART,¬PERFECT)  -->
                          VERBPH(PRM,PERFECT,VFORM=VFORM(VERB))
37   VERBPH(¬PRM,TRANS*,¬OBJECT)  NOUNPH  -->
                                       VERBPH(OBJECT=NOUNPH)
38   NOUNPH  VERBPH(NUMB.EQ.NUMB(NOUNPH),¬SUBJECT)  -->
                             VERBPH(PRM,SUBJECT=NOUNPH,-NUMB)
39   .  /#  -->  PERIOD
40   ./  VERBPH(SUBJECT,OBJECT|¬TRANS*|PASSIVE)  PERIOD  -->
                                       SENT(%VERBPH,-PRM)
```

Semology for decoding English:

```
41   SENT($'ACTION')  -->  ACTSENT(%SENT)
42   ACTSENT(¬PASSIVE)  -->  ACT(%ACTSENT,AGENT=SUBJECT,
                                 GOAL=OBJECT,-SUBJECT,-OBJECT)
43   ACTSENT(PASSIVE)  -->  ACT(%ACTSENT,GOAL=SUBJECT,
                                 -SUBJECT)
44   ACT  -->  OK(%ACT,LC(MEM)=LC(MEM)+1,@LC(MEM)(MEM)=SEG)
45   OK  -->  NULL
```

Figure 4.1. A Set of English Decoding Rules
(cont'd from previous page)

Indicators:

```
SUFX  1-10
   NSFX  1,   E  2,   S  3,   ES  4
   ING  5,   ED  6,   ER  7,   EN  8
TRANS  12
VFORM  13-18
   NUMB  14-15,   SING  14,   PLUR  15
   INFIN  16,   PRESPART  17,   PASTPART  18
PASSIVE  25,   PROG  26,   PERFECT  27,   PRM  33
```

Attributes:

```
SUP  1,   NAME  10
```

Named records:

```
ACTIVITY ('ACTION')
AGENT    ('ATTR')
ARRIV    ('EVENT',E,ES,ING,ED,ER)
BE       ('ATTRVERB',ING,EN)
BIG      ('RELSIZ')
BOAT     ('MOBENTY')
BOY      ('PERSON')
DET      (ATTRIB='DETERM')
EVENT    ('ACTION')
GOAL     ('ATTR')
HAV      ('ATTRVERB',E,ING,TRANS)
LOAD     ('ACTIVITY',NSFX,S,ING,ED,ER,TRANS)
MAN      ('PERSON')
RELSIZ   ('RELVAL',ATTRIB='SIZE')
SERVIC   ('ACTIVITY',E,ES,ING,ED,ER,TRANS)
SHIP     ('MOBENTY')
SIZE     ('ATTR')
THE      ('DET')
UNLOAD   ('ACTIVITY',NSFX,S,ING,ED,ER,TRANS)
```

Figure 4.2. Relevant Declarations from Appendices A and B

specification given in parentheses on the right. Of course, as discussed in the previous chapter, both the condition and creation specifications have access to and can manipulate essentially any information in the computer.

1(a). <u>Morphology</u>

The rules in the morphology of Figure 4.1 specify how characters are to be put together to form stems and parts. For example, rule 1 states that if the string "boy" appears in the input, that string can be described by a NOUNS segment record with a SUP attribute pointing to the named record 'BOY'. Rule 13 says that a NOUNS segment which does not have an ES indicator in the named record pointed to by its SUP, followed by an "s", yields a NOUNP segment which is a copy of the NOUNS segment, but which also has a PLUR indicator. The ES indicator is being used here to mean that the plural (or singular, in the case of a verb) is formed by adding "es" rather than "s". Therefore, if the string "boys" appeared in the input, rule 1 would be applied and then rule 13 would be applied. The result would be a NOUNP segment with a SUP of 'BOY' and a PLUR indicator, to describe the string.

In the morphology of Figure 4.1 there is a rule for each stem, whereas in the encoding morphology of Figure 3.7 this was not required because of the provision of part (ii) of the encoding algorithm (Figure 3.9) which made use of named record names. For a very small vocabulary, it would not matter very much, but for a larger vocabulary, having a rule for each stem would require many decoding rules, all of which would be of

similar form. As will be discussed later in this chapter, in the rules for the queuing problem application there is a group of just 12 rules which recognize any stem for which there is a named record and assign the appropriate segment type to it. Basic to their functioning is a call to a FORTRAN routine which performs a lookup operation among the named records. For this initial example, it was felt best to avoid that extra bit of complexity, however.

In the rules that yield a VERBS it can be seen that in addition to giving a value to the SUP attribute of the segment record, the values of indicators 1 through 10 are copied into the segment record from the named record pointed to by the SUP. The declaration of SUFX as a name for indicators 1 through 10 taken as a group is included in Figure 4.2. The names of the individual indicators in that range (NSFX, E, S, ES, ING, ED, ER, and EN) are declared there also. As an example of their use, it can be seen in the same figure that the named record 'ARRIV' has the indicators E, ES, ING, ED, and ER. These indicators copied into a segment record would help to determine which of rules 16 through 22 could be applied. For example, if one of the strings "arrive", "arrives", "arrived", or "arriving" appeared in the input, it would become a VERBP with a SUP of 'ARRIV' and an appropriate indicator, either PLUR, SING, PASTPART, or PRESPART. However, none of the strings "arriv", "arrivs", or "arriven" would become a VERBP. It can be seen that strings such as "the", "men", and "are" which are complete in themselves skip the stem level and immediately become parts.

In passing, it should be noted once again that the names of indicators, as well as most other names, have no intrinsic meaning to the system, but are made up by the rule-writer in such a fashion as to be meaningful to him, to simplify his task.

1(b). Context

It is possible to specify context in a decoding rule. This is done by placing a slash next to a segment type name, and is interpreted as meaning that even though a segment of that type must be present in order to apply the rule, the string covered by that segment is not to be included in the new segment formed by applying the rule. In the lexology of Figure 4.1, rules 28, 29, 30, and 39 furnish examples of right context, and rule 40 left context.

The use of PUNC for context in rules 28-30 is not really necessary, but was included primarily for illustrative purposes. However, its use in those rules does help to avoid creating some extraneous segments. For example, because of the way the decoding algorithm works (as will be discussed shortly), if the string "#boys#" appeared in the input, the substring "boy" would yield a NOUNS by rule 1, which in turn would yield a NOUNP by rule 12. If rule 29 did not have the contextual constituent /PUNC in it, the substring "#boy" would then become a NOUN, which may go on to participate in other rules, such as 31, 32, 37, and 38, but all for naught. However, with the /PUNC in it, rule 29 says that in order for a space followed by a NOUNP to become a NOUN, they must be followed immediately

by a PUNC, where PUNC is defined by rules 26 and 27 to be a period or a space. Therefore, the substring "#boy" of the string "#boys#" would not become a NOUN, but the substring "#boys" would, by applying rules 1, 13, and 29.

Each decoding rule can have at most one left contextual constituent and one right contextual constituent. It is important to realize that in a rule which has context specified, the new segment formed (e.g. the NOUN in rule 29) does not include any contextual constituent (e.g. the PUNC). Therefore, the left side of rule 29 is not equivalent to # NOUNP PUNC. If the rule were written this way the space after a word would be included as part of the word and would not be available to be considered as the beginning of the next word. The left context in rule 40 is needed to ensure that a SENT segment covers an entire sentence, from the character right after the period of the preceding sentence up to and including the period of the current sentence.

1(c). Lexology

The rules in the lexology of Figure 4.1 specify primarily how words are put together to form phrases and sentences. It can be seen from rules 33-38 that a VERBPH begins at the main verb of a sentence and then "spreads out", first on the right to "pick up" the direct object, if any, and then on the left to pick up any auxiliary verbs and the subject. Finally, when a VERBPH covers the whole sentence, it becomes a SENT in rule 40.

It is instructive to note how rules 33-36 participate in the construction of a segment record to describe a verb phrase such as:

#has#been#being#loaded#

By application of rules 23 and 30, "#has" would be described by the following record:

VERB (SUP:'HAV',SING)

By application of rules 9, 21, and 30, "#been" would be described by:

VERB (SUP:'BE',PASTPART)

From rules 9, 22, and 30, "#being" would be:

VERB (SUP:'BE',PRESPART)

And, from rules 7, 20, and 30, "#loaded" would be:

VERB (SUP:'LOAD',PASTPART)

Then, applying rule 33 "#loaded" would become:

VERBPH (SUP:'LOAD',PASTPART)

Applying rule 34 with the VERB segment for "#being" from above, "#being#loaded" would yield:

VERBPH (SUP:'LOAD',PRM,PASSIVE,PRESPART)

Then, by application of rule 35 with the VERB segment for "#been", "#been#being#loaded" would result in

VERBPH (SUP:'LOAD',PRM,PASSIVE,PROG,PASTPART)

Finally, by applying rule 36 with the VERB segment for "#has",

the whole phrase "#has#been#being#loaded" would be described by:

VERBPH (SUP:'LOAD',PRM,PASSIVE,PROG,PERFECT,SING)

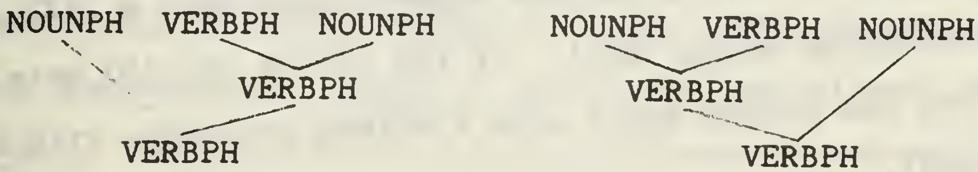
It may be noted that starting with this segment record and applying the encoding rules of Figure 3.7 would result in the creation of the same seven intermediate records shown above (but in a different order, of course) and eventual output of the entire phrase. This was illustrated for the phrase "are being unloaded" in Figure 3.11, the second encoding example.

One of the creation elements which appears in each of rules 34-36 is VFORM=VFORM(VERB). It can be seen in Figure 4.2 that VFORM is declared as the name of indicator bit positions 13-18, which includes SING, PLUR, PRESPART, and PASTPART. So, for example, when rule 35 is applied, the new VERBPH segment record created is a copy of the old one, plus its PRM and PROG indicators are set, plus the values for its indicator bit positions 13-18 are obtained from the corresponding bit positions in the VERB segment record. The effect of this can be seen in the example just given.

The PRM indicator is being used for a different purpose in this set of decoding rules than it was in the set of encoding rules in Figure 3.7. Here PRM stands for "pre modified", and is used to indicate that a VERBPH segment has picked up some elements to the left of the verb which initially created the VERBPH in rule 33. PRM is set by rules 34, 35,

36, and 38, and it is tested by rule 37. (What this amounts to is that if there is a direct object, it must be picked up by the VERBPH, using rule 37, before any auxiliary verbs are.)

PRM is used in this set of rules to prevent multiple identical analyses of a phrase. For example, without PRM in rules 37 and 38, the phrase "#men#load#ships" would result in both of the following two analyses:



These differ only in the order in which the rules are applied and would yield identical segment records for the final VERBPH. With PRM in those rules however, only the first analysis can result.

The use of ATTRIB in rules 28 and 32 may require some explanation. In rule 28 the creation element ATTRIB($(SEG)) results in the value of the ATTRIB attribute of the ADJ segment record being created being obtained from some record in its SUP chain. (This notation was explained in Section A.4(c) of Chapter III.) For example, if the SUP of the ADJ segment were 'BIG', from the declarations in Figure 4.2 it can be seen that the ATTRIB attribute of that segment would get the value 'SIZE' from the named record 'RELSIZ'. ATTRIB is being used here to specify which attribute this ADJ could furnish a value for.

Then there are three conditions given for the application of rule 32. One is that the ADJ segment have an ATTRIB

attribute, the second is that the NOUNPH not already have a value for the attribute specified by the ATTRIB of the ADJ, and the third is that the NOUNPH not already have a DETERMiner. The new NOUNPH created by that rule is a copy of the old one, with one new attribute added, the attribute specified by the ATTRIB of the ADJ, which is made to point to the same named record as the SUP of the ADJ. For example, if the SUP of the ADJ segment were 'BIG', the new NOUNPH would have a SIZE attribute with the value 'BIG'. If the SUP of the ADJ were 'THE', the new NOUNPH would have a DETERM attribute with the value 'THE'.

The string "#the#big#man" would be accepted by two applications of rule 32, but other strings such as "#the#the#man", "#big#the#man", and "#big#big#man" would not be. Clearly, the last of these could be a valid noun phrase, but additional rules would have to be written to treat it properly. For example, "#big#big" could be treated similarly to the phrase "#very#big", or one "#big" could be considered to be the value of the SIZE attribute and the other to be the value of some other attribute such as level-of-importance. This second possibility gets into the area of "multiple word senses", which will be discussed later in this chapter.

Explanations of the condition and creation specifications in rules 37, 38, and 40 may be helpful at this point also. Rule 37 requires a VERBPH that has not been pre-modified (i.e. its PRM indicator is off); it must be a transitive verb (i.e. the named record pointed to by its SUP must have a TRANS

indicator); and it must not already have a direct object (i.e. no OBJECT attribute). The new VERBPH record created would automatically be a copy of the old one, plus it would have an OBJECT attribute whose value would be a pointer to the NOUNPH segment record referred to in the rule.

Rule 38 requires a VERBPH which has the same value for its NUMB indicator (bit positions 14-15) as the NUMB indicator of the preceding NOUNPH (i.e. the verb and potential subject must agree in number); and it must not already have a subject. The new VERBPH record created would automatically be a copy of the old one, plus it would have a PRM indicator, a SUBJECT attribute pointing to the appropriate segment record, and NUMB indicator set to zero.

Rule 40 requires a VERBPH that has a subject. Also, it must either have an object, or be an intransitive verb, or be a passive construction. In addition, it must cover a whole sentence (i.e. have a period at each end). The new segment formed is a SENT which is a copy of the VERBPH. The PRM indicator is turned off because it is no longer needed.

The manner in which a VERBPH is put together in this lexology was decided upon early in this research and used in the rules for the queuing problem application. It seemed like a reasonable approach, primarily because everything else in the VERBPH is considered to depend on the main verb, and, therefore, there could be some advantages to knowing what the verb is before picking up any of the modifiers. However, in doing the rules for the queuing problem application, no

great advantages materialized and it is now felt that a strict left-to-right approach might be better. Certainly, such an approach would have more intuitive appeal, because it is more like what people do when reading English. It should be noted that to change from one approach to the other would simply require rewriting a portion of the rules in the lexology, and would not require modifying either the rule language or the decoding algorithm. Making this change would eliminate the need for the PRM indicator as it is used here.

1(d). Semology

A small semology was included in the decoding rules of Figure 4.1 to illustrate the sort of processing that is typically done at that level. Phrase structure is no longer a consideration, having been handled at the lower levels, so semological rules tend to have only one element on the left side. Rather than being concerned with forming larger segments, these rules are concerned generally with manipulating information in a segment record for an entire sentence, to accomplish some desired task. In the queuing problem application this usually results in information being added to the Internal Problem Description.

The task of the semology in Figure 4.1 is simply to make an action record with AGENT and/or GOAL attributes out of a sentence segment record and store a pointer to it in the MEMORY record. Rule 41 asks if the main verb of the sentence is an action verb by asking if the SENT segment record is in

the set 'ACTION', i.e. is the named record 'ACTION' in its SUP chain. If it is, a copy of the SENT record is made and called an ACTSENT. Then, either rule 42 and 43 is applied, depending upon whether the sentence is active or passive. For an active sentence the AGENT attribute is given the value of the SUBJECT attribute, the GOAL is given the value of the OBJECT (which is zero if there is none), and then the SUBJECT and OBJECT attributes are erased. The values of these attributes are pointers to NOUNPH segment records, as can be seen in rules 37 and 38. For a passive sentence, the GOAL is given the value of the SUBJECT and the SUBJECT is erased.

Rule 44 makes a copy of an ACT produced by one of the two preceding rules and stores a pointer to it in MEMORY. Initially, attribute LC of MEMORY would be 0, so for the first action sentence processed, it would be incremented to 1. Then the creation specification @LC(MEM)(MEM)=SEG would be equivalent to @1(MEM)=SEG, and a pointer to the current record (referred to as SEG) would be stored in attribute 1 of the MEMORY record. As additional action sentences are decoded, pointers to the action records constructed would be stored in attributes 2, 3, etc. of MEMORY.

Rule 45 is needed because of the way the decoding algorithm works. If it were not included, there would be no place for an OK segment to go, so the decoding algorithm would not bother to create one in rule 44, which means that the creation elements in that rule would not be executed. OK and NULL are <u>not</u> special names in this system.

In this simplified semology non-action sentences are ignored. In an expanded set of rules there might be other rules like 41 for other types of sentences, supported by other rules to process the information according to the task at hand. The decoding semology for the queuing problem application has many more rules than this simple example does.

1(e). Counterpart Rules

Comparing the decoding rules of Figure 4.1 with the encoding rules of Figure 3.7, it can readily be seen that many rules in one have a counterpart in the other. Probably the most obvious are pairs like decoding rule 24 and encoding rule 15 in the morphologies:

```
24   A  R  E  -->  VERBP('BE',PLUR)
15   VERBP('BE',PLUR)  -->  A  R  E
```

The first one says that the string "are" yields a VERBP segment with a SUP of 'BE' and a PLUR indicator when decoding, and the second one says that a VERBP segment which has a SUP of 'BE' ad a PLUR indicator yields the string "are" when encoding. Statically speaking, both rules say that "are" is the plural form of "be". It should be realized that in both Figures 3.7 and 4.1 third person and present tense are being assumed.

It is important to remember that because of notational conventions, as described in Section A.4(e) of Chapter III, even though the elements in parentheses in the above two rules are identical, they have different meanings because

they are on a different side in each rule. The first rule has a *creation* specification which says "*set* the SUP attribute to 'BE' and *turn on* the PLUR indicator." The second rule has a *condition* specification which says "*if* the SUP attribute *is* 'BE' and the PLUR indicator *is* on,...."

Another pair of morphological rules which could be considered to be counterparts of one another are decoding rule 21 and encoding rule 18:

```
21   VERBS(EN)  E  N  -->  VERBP(SUP(VERBS),PASTPART)
18   VERBP(PASTPART,EN)  -->  VERBS(SUP(VERBP))  E  N
```

Statically speaking, both of these rules say that the past participle of certain verbs is formed by adding the suffix "en" to the verb stem. In both cases the presence of an EN indicator is required in order to apply the rule. (The verb "be", for example, has an EN indicator associated with it, as can be seen in Figure 4.2.) In the decoding case, the presence of the string "en" at the appropriate place results in the PASTPART indicator being set at the next higher level. In the encoding case, the presence of the PASTPART indicator results in the string "en" at the appropriate place at the next lower level. In both cases, the SUP attribute, which identifies the verb involved, is simply passed to the next level. (For example, the creation element SUP(VERBS) in the first rule means to make the SUP attribute of the segment record being created, i.e. the VERBP segment, be the same as the SUP attribute of the VERBS segment participating in this application of the rule.)

A pair of lexological rules which could be considered to be counterparts are decoding rule 35 and encoding rule 5:

```
35   VERB('BE')  VERBPH(PRESPART,¬PROG,¬PERFECT)  -->
                       VERBPH(PRM,PROG,VFORM=VFORM(VERB))

 5   VERBPH(PROG)  -->  VERB('BE',VFORM=VFORM(VERBPH))
                        VERBPH(-PROG,-VFORM,PRESPART)
```

These rules say that the progressive is formed by having any form of the verb "be" followed by a present participle. Some of the various ways in which this can occur can be seen in the following phrases:

<u>is</u> load<u>ing</u>

<u>is</u> be<u>ing</u> loaded

has <u>be</u>en load<u>ing</u>

has <u>be</u>en be<u>ing</u> loaded

In addition to the three pairs discussed here, there are many other pairs of rules in the two figures that may be considered to be counterparts of one another.

2. <u>The Decoding Algorithm, with Examples</u>

The decoding algorithm is shown in Figure 4.3. Basically what it does, of course, is to apply decoding rules to an input character string in a systematic fashion. In order to apply a rule, the conditions given on the left side of the arrow must be satisfied. This includes the implicit condition that segments of the types specified in the rule must exist and must be contiguous, as well as the explicit conditions specified in parentheses. Applying a rule results in the

(1) Get the next character from the input stream and consider it to be a segment of that type (e.g. an "e" is a segment of type E).

(2) Create a rule instance record for each rule for which this segment can be the first constituent, if there are any such rules, making note of this constituent in each record.

(3) Make a copy of each rule instance record for which this segment can be the next constituent, if there are any such records, making note of this constituent in each new record.

(4) If there is a rule instance record which is complete (i.e. has all of its constituents), create a segment record according to the right side of the associated rule (i.e. apply the rule), erase the rule instance record, and go to step 2.

(5) If the input stream is not empty, go to step 1.

(6) Halt.

Figure 4.3. The Decoding Algorithm

creation of a segment record, usually to describe a larger portion of text (e.g. rule 34 of Figure 4.1), but in some cases just to furnish an additional description of a particular portion of text (e.g. rule 33). Rule application may also affect the longer-term information structure being built (e.g. rule 44).

The decoding algorithm deals with one character at a time from the input stream. After obtaining a character, it essentially applies every rule that can be applied up to that point in the input string, but without redoing applications that have already been done. As will be seen, this does not mean that any particular portion of the string can be described by only one segment record at a time.

2(a). Rule Instance Records

An important part of the decoding algorithm is something called a "rule instance record", which primarily maintains information about the potential applicability of a rule. A rule instance record is initially created for a rule whenever a segment which can be the first constituent of that rule becomes available (step 2). A terminal segment (i.e. a letter, digit, or special character) becomes available by being obtained from the input stream (step 1), and a non-terminal segment becomes available by the application of a rule (step 4). A rule instance record is "extended" whenever a segment which can be the next constituent of the associated rule becomes available (step 3). When a rule instance record finally indicates that

all constituents are available, the associated rule is applied, resulting in a new segment becoming available (step 4).

A rule instance record has the following attributes:

1. identifcation number of the associated rule
2. leftmost position covered by this rule instance record
3. rightmost position covered by this rule instance record
4. number of constituents available so far
5. pointers to constituent segment records

Each rule has an internal identification number which is assigned to it when it is compiled. For illustrative purposes here, the numbers appearing with the rules in Figure 4.1 will be regarded as their identification numbers. Each character in the input string is considered to have a position number associated with it, with successive numbers being assigned as characters are obtained from the input stream.

As an example of the use of rule instance records, if the string "boys" were in the input stream beginning at position 5, one of the rule instance records which would be created with the "b" is obtained is:

T1 (1,5,5,1,0)

where the numbers separated by commas are the values of the five attributes listed above, in order. This particular record carries the information that a segment which can be

the first constituent (attribute 4) of rule 1 (attribute 1) exists at position 5 (attributes 2 and 3). Attribute 5 has the value 0 because the first constituent is a terminal segment, and therefore has no segment record associated with it.

Then when the "o" at position 6 is obtained, rule instance record T1 shown above would be extended to produce the record:

T2 (1,5,6,2,0,0)

This record carries the information that segments which can be the first 2 constituents of rule 1 cover positions 5 through 6, and that these segments have no associated segment records.

Then when the "y" at position 7 is obtained, the record T2 would be extended to produce the record:

T3 (1,5,7,3,0,0,0)

Because the value of attribute 4 of this record is the same as the number of constituents in the associated rule (i.e. rule 1 has 3 constituents), this rule instance record would be considered to be "complete" (i.e. it has all of its constituents). At this point rule 1 can be applied to yield a NOUNS segment with a SUP of 'BOY', covering positions 5-7. This information can be summarized as:

NOUNS 5-7 R1 (SUP:'BOY')

where "R1" is being used to identify the segment record created by application of the rule.

One of the rule instance records which would then be

created as a result of this NOUNS segment becoming available is:

T4 (13,5,7,1,R1)

This record carries the information that a segment which can be the first constituent of rule 13 covers positions 5-7 and this segment is described by segment record R1. It should be noted that this rule instance record would not have been created if the condition specified in parentheses with NOUNS in rule 13 were not satisfied.

Finally for this example, when the "s" at position 8 is obtained, the record T4 would be extended to produce:

T5 (13,5,8,2,R1,0)

Because rule 13 has 2 constituents, this record is complete, and rule 13 can be applied to yield the segment:

NOUNP 5-8 R2 (SUP:'BOY',PLUR)

There are no rules beginning with NOUNP, so this would not result in the creation of any new rule instance records. However, if the string "boys" had been preceded by a "#" at position 4, a rule instance record for rule 29 would have been created at that time, and now it could be extended to:

T6 (29,4,8,2,0,R2)

It should be noted that other rule instance records in addition to T1-T5 shown above would have been created during the processing of the string "boys", but they were not shown, in order to keep the example simple. When the "b" was obtained,

records would have been created for rules 2, 9, and 10, also, because they begin with B. Then when the "o" was obtained, the record for rule 2 would have been extended too. Also, when the NOUNS segment became available, a record would have been created for rule 12, in addition to the one shown for rule 13. This one would have been complete immediately to yield:

NOUNP   5-7   R3 (SUP:'BOY',SING)

Similarly to what was said in the preceding paragraph, this could also have resulted in an extension of a rule instance record for rule 29. (It can be infered from the copying in step 3 of the decoding algorithm that one rule instance record may furnish the basis for any number of extensions. Another example of this will be seen in the next subsection.)

2(b). First Example

Figure 4.4 is an example of the application of the decoding algorithm of Figure 4.3 using the set of rules given in Figure 4.1, supported by the declarations given in Figure 4.2. This figure shows in detail the processing of the first part of the sentence "Men are arriving." It shows each segment as it becomes available and each rule instance record. A later figure shows just the segments, but for the whole sentence.

When the decoding routine is initially called, a period and a space are inserted into the input stream just prior to any actual input, to simulate the end of a previous sentence.

Step		Rule instance records	Rule	Consts	Seg	Type	Pos	Record
1					1	.	1	
2		T1 (26,1,1,1,0)						
2		T2 (39,1,1,1,0)						
2		T3 (40,1,1,1,0)						
4			26	1	2	PUNC	1-1	R1 ( )
1					3	#	2	
2		T4 (27,2,2,1,0)						
2		T5 (28,2,2,1,0)						
2		T6 (29,2,2,1,0)						
2		T7 (30,2,2,1,0)						
3	T2 →	T8 (39,1,1,2,0,0)						
4			27	3	4	PUNC	2-2	R2 ( )
4			39	1/3	5	PERIOD	1-1	R3 ( )
1					6	M	3	
2		T9 (14,3,3,1,0)						
2		T10 (15,3,3,1,0)						
1					7	E	4	
3	T9 →	T11 (14,3,4,2,0,0)						
1					8	N	5	
3	T11 →	T12 (14,3,5,3,0,0,0)						
4			14	6-8	9	NOUNP	3-5	R4 (SUP:'MAN',PLUR)
3	T6 →	T13 (29,2,5,2,0,R4)						
1					10	#	6	
2		T14 (27,6,6,1,0)						
2		T15 (28,6,6,1,0)						
2		T16 (29,6,6,1,0)						
2		T17 (30,6,6,1,0)						
4			27	10	11	PUNC	6-6	R5 ( )
3	T13 →	T18 (29,2,5,3,0,R4,R5)						
4			29	3,9/11	12	NOUN	2-5	R6 (SUP:'MAN',PLUR)
2		T19 (31,2,5,1,R6)						
4			31	12	13	NOUNPH	2-5	R7 (SUP:'MAN',PLUR)
2		T20 (38,2,5,1,R7)						

Figure 4.4. First Decoding Example, Showing Rule Instance Records and Segments

The input stream given to the decoding algorithm for this example then is:

. # m e n # a r e # a r r i v i n g . #

The algorithm begins at step 1 by getting the initial period from the input stream, and considering it to be a segment of that type. The first line in Figure 4.4 shows that this is the first segment to become available and it is at (i.e. it covers) position 1. The next three lines show the result of performing step 2. A rule instance record is created for each of the three rules that begin with a period. Step 3 produces nothing this time because there are no rule instance records in existence for which this period can be the next constituent.

Then at step 4 rule instance record T1 is seen to be complete, so rule 26 is applied, yielding the PUNC segment shown. Because there is no creation specification in rule 26, the segment record created (R1) has no attributes. The column labelled "Consts" shows which segments are the constituents when a rule is applied, with a segment being referred to by the number it has in the Seg column. In this case the constituent of rule 26 is segment 1, the initial period. Then record T1 is erased, and the algorithm returns to step 2, this time to process the PUNC segment which just became available. Steps 2, 3, and 4 produce nothing this time, however, because there are no rules which have PUNC as the first constituent, there are no rule instance records in existence

for which this PUNC can be the next constituent, and there are no complete rule instance records.

Back at step 1, the next character (#) is obtained from the input stream and considered to be segment 3. Because there are four rules beginning with #, four rule instance records are created at step 2 (T4-T7). Then at step 3, record T2 is extended to produce T8, because a # at position 2 can be the next constituent of T2. The rightmost position covered by T8 (the value of its third attribute) is 1 rather than 2 because the # is a contextual constituent in rule 39.

Conceptually, the search in step 3 of the algorithm requires examining all rule instance records which have not been erased. In order to extend a rule instance record with a segment which has just become available, three conditions must be met:

1. The rightmost position covered by the rule instance record (attribute 3) must be 1 less than the leftmost position covered by the available segment.

2. The segment type of the next (attribute 4 plus 1) constituent of the associated rule (attribute 1) must be the same as the type of the available segment.

3. Any conditions specified in parentheses with that constituent in the rule must be satisfied.

At step 4, a search for complete rule instance records begins at T4 this time (because T1-T3 were already examined at earlier iterations), and record T4 is found to be complete. As a result, rule 27 is applied, yielding the PUNC segment 4. Then T4 is erased and the algorithm returns to step 2, but

neither step 2 nor step 3 produce anything. At step 4 again records T5-T8 are examined and T8 is found to be complete. Application of rule 39 then yields the PERIOD segment 5. The slash in the Consts column indicates context. After erasing T8, the algorithm again returns to step 2. This time nothing is produced by steps 2, 3, or 4, so the algorithm goes back to step 1 to get the next character (m) from the input stream.

Processing continues in this manner until the input stream is empty. The details of this processing for the next four characters (men#) are shown in the remainder of Figure 4.4. Figure 4.5 shows the segment portion of the information produced during processing for the whole sentence, with the first 13 lines being identical to the right side of Figure 4.4.

The overall progress of the decoding process for this example sentence can be followed with the information that is given in Figure 4.5. It can be seen that after the segments M, E, and N are obtained from the input stream in lines 6-8, rule 14 is applied to yield a NOUNP in line 9. Then the # is obtained, yielding a PUNC in line 11, which furnishes the right-context needed to apply rule 29 with segments 3 and 9, to yield a NOUN in line 12. This in turn serves as the constituent for rule 31, to result in a NOUNPH at line 13. The details up to this point can be seen in the previous figure. The last rule instance record (T20) shown there is for rule 38, and may be interpreted as saying that this NOUNPH is being considered as a potential subject.

Rule	Consts	Seg	Type	Pos	Record	
		1	.	1		
26	1	2	PUNC	1-1	R1	( )
		3	#	2		
27	3	4	PUNC	2-2	R2	( )
39	1/3	5	PERIOD	1-1	R3	( )
		6	M	3		
		7	E	4		
		8	N	5		
14	6-8	9	NOUNP	3-5	R4	(SUP:'MAN',PLUR)
		10	#	6		
27	10	11	PUNC	6-6	R5	( )
29	3,9/11	12	NOUN	2-5	R6	(SUP:'MAN',PLUR)
31	12	13	NOUNPH	2-5	R7	(SUP:'MAN',PLUR)
		14	A	7		
		15	R	8		
		16	E	9		
24	14-16	17	VERBP	7-9	R8	(SUP:'BE',PLUR)
		18	#	10		
27	18	19	PUNC	10-10	R9	( )
30	10,17/19	20	VERB	6-9	R10	(SUP:'BE',PLUR)
33	20	21	VERBPH	6-9	R11	(SUP:'BE',PLUR)
38	13,21	22	VERBPH	2-9	R12	(SUP:'BE',SUBJECT:R7,PRM)
		23	A	11		
		24	R	12		
		25	R	13		
		26	I	14		
		27	V	15		
4	23-27	28	VERBS	11-15	R13	(SUP:'ARRIV',E,ES,ING,ED,ER)
		29	I	16		
		30	N	17		
		31	G	18		
22	28-31	32	VERBP	11-18	R14	(SUP:'ARRIV',PRESPART)
		33	.	19		
26	33	34	PUNC	19-19	R15	( )
30	18,32/34	35	VERB	10-18	R16	(SUP:'ARRIV',PRESPART)
33	35	36	VERBPH	10-18	R17	(SUP:'ARRIV',PRESPART)
35	20,36	37	VERBPH	6-18	R18	(SUP:'ARRIV',PRM,PROG,PLUR)
38	13,37	38	VERBPH	2-18	R19	(SUP:'ARRIV',SUBJECT:R7,PRM,PROG)
		39	#	20		
27	39	40	PUNC	20-20	R20	( )
39	33/39	41	PERIOD	19-19	R21	( )
40	1/38,41	42	SENT	2-19	R22	(SUP:'ARRIV',SUBJECT:R7,PROG)
41	42	43	ACTSENT	2-19	R23	(SUP:'ARRIV',SUBJECT:R7,PROG)
42	43	44	ACT	2-19	R24	(SUP:'ARRIV',AGENT:R7,PROG)
44	44	45	OK	2-19	R25	(SUP:'ARRIV',AGENT:R7,PROG)
					MEM	(@1:R25)

Figure 4.5. First Decoding Example, Showing Just Segments

Then the A, R, and E are obtained, yielding a VERBP in line 17. The # in line 18 results in a PUNC, which provides the context needed to produce a VERB in line 20. At this point step 2 of the algorithm would produce the following three rule instance records (where the identifying numbers with the T are being selected arbitrarily):

T31 (33,6,9,1,R10)
T32 (34,6,9,1,R10)
T33 (35,6,9,1,R10)

By looking at the rules associated with each of these records, it can be seen that what this means is that this VERB is being considered as possibly being the main verb (rule 33) or being an auxiliary verb in a passive (rule 34) or progressive (rule 35) construction.

Record T31 is immediately complete because rule 33 has just one constituent, and it produces the VERBPH segment 21. With this segment available, record T20 (shown in Figure 4.4) can be extended to yield:

T34 (38,2,9,2,R7,R11)

This rule instance record is complete and results in the VERBPH segment 22, which covers positions 2-9 (#men#are).

As it turns out, this is not the desired analysis of that portion of the input. However, it does not matter, because segments 13 and 20, which cover positions 2-5 and 6-9 respectively, are still available. Also, record T20 still exists and could be extended differently later. In

general, segments are <u>not</u> "replaced" by other segments during the processing, but rather, multiple partial analyses may exist simultaneously. Hopefully, only one analysis will eventually cover the whole sentence, however.

The processing of "arriving.", resulting in a VERB segment covering positions 10-18, is shown in lines 23-35. Application of rule 33 then results in a VERBPH segment covering positions 10-18 in line 36. The availability of this segment makes it possible to extend record T33 shown earlier to get the record:

T41 (35,6,18,2,R10,R17)

Since this record is complete, rule 35 is applied to yield the VERBPH segment covering 6-18 shown in line 37. Then with this segment available, record T20 can be extended <u>again</u> to get:

T42 (38,2,18,2,R7,R18)

This record is complete, so rule 38 is applied to yield the VERBPH segment covering 2-18 shown in line 38.

Four lines later a SENT segment is formed, then an ACTSENT, then an ACT, and finally an OK in line 45. As can be seen there, the associated segment record (R25) has a SUP of 'ARRIV', an AGENT attribute pointing to segment record R7, and a PROG indicator. Record R7 can be seen in line 13 to have a SUP of 'MAN' and a PLUR indicator. Also shown in the figure is the MEMORY record with its attribute 1 pointing to record R25. This value was set by the application of rule 44.

All records other than MEMORY, R25, and R7 can be discarded at this point. The manner in which this is done is described in Subsection 4 which deals with the computer implementation.

What has been accomplished by the processing shown for this example is that the information contained in the character string

"men are arriving."

has been converted into equivalent information in the form of the records

R25 (SUP:'ARRIV',AGENT:R7,PROG)

R7 (SUP:'MAN',PLUR)

2(c). Second Example

Figure 4.6 furnishes another example of the operation of the decoding algorithm. The input sentence "The men unload the big ships." is the same sentence that was used for an example in Figures 3.2, 3.4, 3.6, and 3.10 in Chapter III. The format of Figure 4.6 is essentially the same as that of Figure 4.5, but it has less detail. The input string appears on the left side of the figure, with the position number of each character given beside it. This part of the figure is not intended to be lined up in any way with the other part.

Although the right part of Figure 4.6 is in the same format as Figure 4.5, it does not show all of the segments produced during the processing of the input sentence. Only those non-terminal segments which appear in later rule

Inp	Pos	Rule	Consts	Seg	Type	Pos	Record	
#	2							
T	3	11	T-E	1	ADJP	3-5	R1	(SUP:'THE')
H	4	28	#,1	2	ADJ	2-5	R2	(SUP:'THE',ATTRIB:'DETERM')
E	5	14	M-N	3	NOUNP	7-9	R3	(SUP:'MAN',PLUR)
#	6	29	#,3	4	NOUN	6-9	R4	(SUP:'MAN',PLUR)
M	7	31	4	5	NOUNPH	6-9	R5	(SUP:'MAN',PLUR)
E	8	32	2,5	6	NOUNPH	2-9	R6	(SUP:'MAN',DETERM:'THE',PLUR)
N	9	6	U-D	7	VERBS	11-16	R7	(SUP:'UNLOAD',NSFX,S,ING,ED,ER)
#	10	16	7	8	VERBP	11-16	R8	(SUP:'UNLOAD',PLUR)
U	11	30	#,8	9	VERB	10-16	R9	(SUP:'UNLOAD',PLUR)
N	12	33	9	10	VERBPH	10-16	R10	(SUP:'UNLOAD',PLUR)
L	13	11	T-E	11	ADJP	18-20	R11	(SUP:'THE')
O	14	28	#,11	12	ADJ	17-20	R12	(SUP:'THE',ATTRIB:'DETERM')
A	15	10	B-G	13	ADJP	22-24	R13	(SUP:'BIG')
D	16	28	#,13	14	ADJ	21-24	R14	(SUP:'BIG',ATTRIB:'SIZE')
#	17	3	S-P	15	NOUNS	26-29	R15	(SUP:'SHIP')
T	18	13	15,S	16	NOUNP	26-30	R16	(SUP:'SHIP',PLUR)
H	19	29	#,16	17	NOUN	25-30	R17	(SUP;'SHIP',PLUR)
E	20	31	17	18	NOUNPH	25-30	R18	(SUP:'SHIP',PLUR)
#	21	32	14,18	19	NOUNPH	21-30	R19	(SUP:'SHIP',SIZE:'BIG',PLUR)
B	22	32	12,19	20	NOUNPH	17-30	R20	(SUP:'SHIP',SIZE:'BIG',DETERM:'THE',PLUR)
I	23	37	10,20	21	VERBPH	10-30	R21	(SUP:'UNLOAD',OBJECT:R20,PLUR)
G	24	38	6,21	22	VERBPH	2-30	R22	(SUP:'UNLOAD',SUBJECT:R6,OBJECT:R20,PRM)
#	25	40	22,.	23	SENT	2-31	R23	(SUP:'UNLOAD',SUBJECT:R6,OBJECT:R20)
S	26	41	23	24	ACTSENT	2-31	R24	(SUP:'UNLOAD',SUBJECT:R6,OBJECT:R20)
H	27	42	24	25	ACT	2-31	R25	(SUP:'UNLOAD',AGENT:R6,GOAL:R20)
I	28	44	25	26	OK	2-31	R26	(SUP:'UNLOAD',AGENT:R6,GOAL:R20)
P	29						MEM	(@1:R26)
S	30							
.	31							

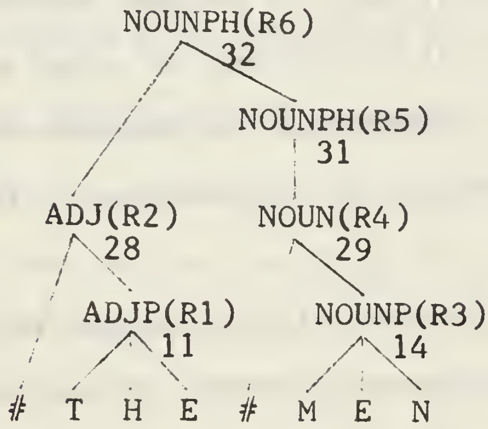
Figure 4.6. Second Decoding Example

applications are included. In other words, "dead-end" partial analyses like segments 21 and 22 in Figure 4.5 are not shown. PUNC segments, which serve only as contextual constituents, have also been left out. The input string and position numbers given in the left part of the figure furnish all of the information needed about terminal segments. In the Consts column, terminal segments are referred to by name and non-terminal segments are referred to by number (the number in the Seg column).

The example in Figure 4.6 is intended to demonstrate the manner in which the record structure is built up for a sentence of a bit more complexity than the previous example. The particular sentence used was chosen to make it possible to more easily relate the material of this chapter to that of Chapter III.

It can be seen in the figure that initially rule 11 is applied to the segments T-E in positions 3-5 to yield an ADJP with a SUP of 'THE'. Then rule 28 results in an ADJ with a SUP of 'THE' and an ATTRIB of 'DETERM' covering positions 2-5. Similarly, rules 14 and 29 yield a NOUN with a SUP of 'MAN' and a PLUR indicator, covering positions 6-9. This NOUN becomes a NOUNPH (segment 5), which then picks up the ADJ (segment 2) by application of rule 32 to form a new NOUNPH (segment 6) covering positions 2-9. This new NOUNPH has a SUP of 'MAN', a DETERM of 'THE', and a PLUR indicator, as can be seen in line 6.

Another way of displaying exactly the same information as appears in the first six lines of Figure 4.6 is the following, where the numbers shown are rule numbers:




```
 R1   (SUP:'THE')
 R2   (SUP:'THE',ATTRIB:'DETERM')
R3-5  (SUP:'MAN',PLUR)
 R6   (SUP:'MAN',DETERM:'THE',PLUR)
```

This drawing can be seen to be basically the same as the leftmost portion of the "tree" drawn to show the phrase structure of this sentence in Figure 3.2. The rule numbers are different simply because different sets of rules were used (Figure 3.1 vs. Figure 4.1). What is significantly different about this drawing, however, is that each point in the tree has a record associated with it containing information which is considered to be equivalent to that contained in the entire sub-tree beneath that point.

If the next several lines of Figure 4.6 are compared with Figure 3.2, it can be seen that the decoding algorithm actually goes through the same series of steps which would be required to build that whole tree, in sort of a left-to-right, bottom-

to-top fashion. The algorithm does not explicitly build the tree, however, because at each point all of the necessary information about the sub-tree beneath that point is contained in the segment record, and the segment records are the objects referred to in the rules. It may be noted that after line 21 the correspondence between the two figures breaks down slightly, but that is only because of differences in the two sets of rules.

The last few lines of Figure 4.6 are similar to those of Figure 4.5. In this case, however, there is an OBJECT attribute present which then becomes a GOAL attribute. The following records may be considered to be the final result of decoding the sentence of this example:

```
MEM (@1:R26)
R26 (SUP:'UNLOAD',AGENT:R6,GOAL:R20)
R6  (SUP:'MAN',DETERM:'THE',PLUR)
R20 (SUP:'SHIP',SIZE:'BIG',DETERM:'THE',PLUR)
```

Except for the DETERM attributes, these last three records are the same as the three records which were available at the start of the encoding example given in Figure 3.10. That example showed the encoding of the same sentence.

It should be pointed out once more that Figure 4.6 does not show all of the segments created during the processing of this sentence, but only the ones involved in the "correct" analysis, i.e. the analysis which eventually results in an OK. As an example of one that is not shown, at some point rule 38 would have also been applied using NOUNPH segment 5 and VERBPH

segment 10 to yield:

VERBPH 6-16 (SUP:'UNLOAD',SUBJECT:R5,PRM)

This segment would not be able to be combined with any others, but its presence during the processing is inconsequential.

3. <u>More About the Decoding Rule Language</u>

It has already been seen that the decoding rule language is basically the same as the encoding rule language. All of the material on the language of specification elements in Section A.4 of Chapter III and on the execution of specification elements in Section A.6 of that chapter is equally applicable to decoding rules. There are some additional points which need to be discussed here however.

3(a). <u>Naming Conventions</u>

Because a decoding rule usually has more than one segment type (i.e. constituent) on the left side, the idea of using a segment type name as a pointer value to refer to a segment on the left needs to be expanded somewhat from what was stated in subsection A.4(c) of the previous chapter. In a decoding rule a segment type name may be used in a condition specification to refer to a segment further to the left in the rule. An example of this can be seen in rule 38 of Figure 4.1, where the indicator designator NUMB(NOUNPH) is used.

There is also the reference problem associated with having more than one segment of the same type on the left side of a rule. In this case a #n, where n is a digit, can

be appended to the segment type name to distinguish among them. For example, application of the rule

NOUN NOUN --> NOUNPH(%NOUN#2,MOD=NOUN#1)

would result in a NOUNPH which is a copy of the second NOUN segment and which has a pointer to the first NOUN segment as the value of its MOD attribute. Actually, the same result would be achieved if the #1 were left off.

As has been seen in the examples, the "automatic copy" convention applies to decoding rules also. When the segment type on the right of the arrow has the same name as some segment type on the left, the segment record created begins as a copy of the segment record referred to on the left. If there is more than one segment type on the left with the same name, then the first one is used. For example, application of the rule

NOUN NOUN --> NOUN(   )

would result in a NOUN segment which begins as a copy of the first NOUN segment referred to on the left. Of course, the convention can be overridden by using an explicit copy specification element:

NOUN NOUN --> NOUN(%NOUN#2,   )

3(b). <u>The Segment-Types LETTER, DIGIT, and ENCODING</u>

Two segment types which are specially defined for use on the left side of decoding rules are LETTER and DIGIT. In addition to being a segment of its own type, each letter

appearing in the input string is considered to be a segment of type LETTER (e.g. an "e" is both an E and a LETTER), and each digit is considered to be a segment of type DIGIT (e.g. a "4" is both a 4 and a DIGIT). As far as the user is concerned, the effect of having these is exactly the same as though the following 36 rules were included in the morphology:

```
A  -->  LETTER
:
Z  -->  LETTER

0  -->  DIGIT
:
9  -->  DIGIT
```

However, by having these two segment types specially defined, they can be processed more efficiently by the system. Examples of the use of LETTER and DIGIT can be seen in the decoding morphology for the queuing problem application in Appendix I.

Another specially defined segment type for use in decoding rules is ENCODING. This can be used on the right side of a decoding rule to result in the encoding routine being called from the decoding routine when that rule is applied. For example, if a question is decoded, this could be used to initiate processing to produce the answer. When the encoding routine is called in this way, the segment type ENCODING is put on the stack to begin, along with a pointer to the segment record which was just created by applying the decoding rule. Encoding takes place in the usual manner, and then when the stack finally becomes empty, decoding resumes.

As a simple illustration of the use of ENCODING, the following could be used to "link" the decoding rules of Figure 4.1 and the encoding rules of Figure 3.7:

<u>Semology for decoding English</u>:

OK --> ENCODING(%OK)

ENCODING --> NULL

<u>Semology for encoding English</u>:

ENCODING --> SENT(%ENCODING)

If the system were given all of the information from Figures 4.2, 4.1, and 3.7, plus the above, and then given an action sentence to decode, it would decode the sentence and then immediately encode it.

For example, if it were given the sentence "The men unload the big ships",it would go through exactly the same analysis summarized in Figure 4.6, with the following segment added at the end:

27 ENCODING 2-31 R27 (SUP:'UNLOAD',AGENT:R6,GOAL:R20)

As soon as this segment is created, the encoding routine would be called with ENCODING(R27) on the stack. Then this would come off the stack, and the semological encoding rule given above would be applied according to step (i) of the encoding algorithm to put SENT(R28) on the stack, where R28 would be a copy of R27 (which was a copy of R26). It can be seen that this is the same information situation that existed at the beginning of the encoding example given in Figure 3.10, and

would result in the output "The men unload the big ships."

If it were given the sentence "Men are arriving.", it would produce the output "The men are arriving.", because the encoding rules of Figure 3.7 always insert a "the" at the beginning of a noun phrase. If the semological encoding rule shown above were changed to

ENCODING --> SENT(%ENCODING,PASSIVE,PROG)

and then the system were given the sentence "The men unload the big ships.", the resulting output would be "The big ships are being unloaded." It should be noted that because of the record copying involved, the encoding being done would not affect the information structure being built (i.e. the records pointed to from MEMORY would remain intact). The use of ENCODING in the queuing problem application can be seen in Appendices H and I.

3(c). Additional Features

Step 2 of the decoding algorithm given in Figure 4.3 says, "Create a rule instance record for each rule for which this segment can be the first constituent...." However, in many cases, especially in the semology, it would be more convenient if that step said, "Create a rule instance record for the first rule for which this segment can be the first constituent...." Therefore, provisions have been made to have the decoding algorithm work this way for segment types so designated by the user. As an example of this, the following

Once the rules have been entered into the system, decoding can be initiated by the command

DECODE:

All input after that is considered to be in the input stream for the decoding routine, until a line ending in a colon is encountered. This may be another command or simply an end-of-file marker.

The discussion of the decoding algorithm given earlier was kept free of implementation details to make the main points clearer. Some details about the information structure and the decoding routine are presented here.

4(a). <u>Segment-Type Records</u>

A discussion of "segment-type records" and the role they play in encoding was given in Section A.3(d) of Chapter III. These same records play a similar role in decoding. In addition to having an attribute with a pointer to the first encoding rule that has that segment type on the left, such a record also has an attribute with a pointer to the first decoding rule that begins with that segment type. Each decoding rule then has a pointer to the next rule beginning with the same segment type. This has the effect of partitioning the set of rules into subsets according to their first constituents, and forming a linked list for each subset. Whenever step 2 of the decoding algorithm is executed, the list for the current segment type is traversed.

During decoding, each segment-type record also has an

This rule would create two segment records to describe the string "shop", and would be equivalent to, but more efficient than, the pair of rules:

```
S  H  O  P  -->  NOUNS('SHOP1')
S  H  O  P  -->  VERBS('SHOP2',SUFX*)
```

Some of the rules in Appendix I have more than one segment on the right side.

There are two more points which should be mentioned here. One is that if contiguous spaces appear in the input stream, all but the first are ignored. This means, for example, that the input "men###are##arriving.#" would be treated identically to the input "men#are#arriving.#"

The final point is that the left side of a rule may consist merely of a contextual constituent with no other constituents. This results in a segment which covers no text, but may be thought of as existing between two positions. Optional elements can sometimes be conveniently handled this way. Some rules of this sort can be seen in Appendix I also.

4. <u>Computer Implementation</u>

Decoding rules are compiled by the same FORTRAN routine which compiles encoding rules, so the manner of entering them into the system is the same as that described in Section A.5 of Chapter III. Of course, in this case the heading cards contain the word "decoding" instead of "encoding". The internal format of decoding rules is basically the same as described for encoding rules in that same section of the previous chapter also.

Once the rules have been entered into the system, decoding can be initiated by the command

DECODE:

All input after that is considered to be in the input stream for the decoding routine, until a line ending in a colon is encountered. This may be another command or simply an end-of-file marker.

The discussion of the decoding algorithm given earlier was kept free of implementation details to make the main points clearer. Some details about the information structure and the decoding routine are presented here.

4(a). Segment-Type Records

A discussion of "segment-type records" and the role they play in encoding was given in Section A.3(d) of Chapter III. These same records play a similar role in decoding. In addition to having an attribute with a pointer to the first encoding rule that has that segment type on the left, such a record also has an attribute with a pointer to the first decoding rule that begins with that segment type. Each decoding rule then has a pointer to the next rule beginning with the same segment type. This has the effect of partitioning the set of rules into subsets according to their first constituents, and forming a linked list for each subset. Whenever step 2 of the decoding algorithm is executed, the list for the current segment type is traversed.

During decoding, each segment-type record also has an

attribute with a pointer to the first rule instance record "waiting for" a segment of this type. Each rule instance record then has a pointer to the next one waiting for a segment of the same type. This has the effect of partitioning the set of incomplete rule instance records at any time into subsets according to their next constituents, and forming a linked list for each subset. Whenever step 3 of the decoding algorithm is executed, the list for the current segment type is traversed. These lists are dynamic, growing and shrinking during decoding, whereas the lists described in the previous paragraph are static, being established at rule compilation time.

4(b). Rule Instance Records

Rule instance records are stored in a separate array, with each record consisting of a series of contiguous two-byte elements. Each of these elements holds the value of one attribute. In addition to the attributes shown earlier, a rule instance record also has a link attribute which points to the next rule instance record on some list (or is zero if this is the last one on the list). There are the various lists of incomplete rule instance records just described, and there may be a list of complete rule instance records at any time. Also, there is a list of records that are available to be used when a rule instance record is to be created.

In an earlier version of this system, rule instance records were stored in the CELL array, just as other records are.

Recently however, to take advantage of their special form the separate array was set up for them. This resulted in reducing memory requirements for these records by about 75 percent and decoding time by almost 50 percent.

4(c). <u>The Decoding Routine</u>

When the decoding routine is called in response to the DECODE command, prior to executing step 1 of the algorithm, some initialization is performed. This includes putting a period and space at the beginning of the input stream and setting the position counter to zero. Also, the rule instance record array is set up as a list of available records.

At step 1 a pointer to the appropriate segment-type record is obtained from a 256-element vector, using the EBCDIC representation of the input character as an index. For example, the decimal equivalent of the EBCDIC representation for the letter "e" is 197, and element 197 of this vector is a pointer to the segment-type record for E. As described above, this segment-type record furnishes access to the information needed to execute steps 2 and 3.

At step 2 the list of rules beginning with a segment of the current type is traversed. For each rule the condition specification with the first constituent is executed by a call to CRSEG, in the manner described in Section A.6 of Chapter III, and if it is satisfied (or if there is no condition), a rule instance record is created. This is accomplished by getting a record from the list of those available and storing

appropriate values in its attributes. If this rule instance record is complete (i.e. has all of its constituents), it is added to the list of complete records. Otherwise, it is added to the list of incomplete records associated with the segment type of the <u>next</u> constituent in the rule, i.e. it is waiting for a segment of that type.

At step 3 the list of incomplete rule instance records waiting for a segment of the current type is traversed. For each record the rightmost position covered is compared to the leftmost position of the current segment to see if they are adjacent. If they are, the condition specification with the next constituent of the associated rule is executed. If it is satisfied (or if there is none), another rule instance record is created by getting a record from the list of those available. The attributes of this new record are set equal to those of the other one, and then the number of constituents is increased by 1, and a pointer to the current segment record (if any) is stored in the appropriate attribute. Also, the value of the rightmost position attribute is set properly. If this rule instance record is complete, it is added to the list of complete records. Otherwise, it is added to the list of incomplete records waiting for a segment of the type of the next constituent in the associated rule.

At step 4 if the list of complete rule instance records is not empty, the first record is taken off the list and the associated rule is applied. Application of the rule results in a new current segment type, with a pointer to the appropriate

segment type record being obtained from the internal representation of the rule. The creation specification is executed by a call to CRSEG, which has access to the segments referred to on the left of the rule through the pointers stored in the rule instance record. Then the record is erased, i.e. put back on the list of available rule instance records. Control can then go back to step 2 with the new current segment type which was just established here.

At step 5 if the space after a word or if the period or question mark at the end of a sentence has just been processed, a "purging" operation is performed to remove certain rule instance records from the lists of incomplete records and erase them, i.e. return them to the list of available records. When the character just processed is a space, all rule instance records waiting for a morphological segment that begins at or prior to the position occupied by the space can be purged. These are records that could not possibly be extended, because morphological segments do not cross word boundaries. Similarly, when the character just processed is a period or question mark, all rule instance records waiting for a lexological segment that begins at or prior to the position occupied by that character can be purged. These are records that could never be extended because lexological segments do not cross sentence boundaries. After purging, control returns to step 1 if the input stream is not empty.

At step 6 all rule instance records are purged, and a return is made from the decoding routine.

Actually at step 3 if the current segment is a terminal segment, rather than making a copy of the rule instance record which is being extended, the same record can be used. It is removed from one list, modified, and added to another list. Because a terminal segment type has a unique realization, there is no way that the original record might be needed to be extended again.

Although it is not currently being done, a one-character look-ahead could be employed in both steps 2 and 3 to avoid keeping a rule instance record to wait for a terminal segment that is not there. This would eliminate the need for the morphological purge at step 5.

4(d). Segment Records

It has been seen that a segment record is created at step 4. Then in steps 2 and 3 pointers to that segment record may be stored in rule instance records. Later, pointers to that segment record may also be stored in other segment records, or named records, or MEMORY. If it turns out that a segment record is not being pointed to, its reference counter would be 0, and the record can be erased. The purging of rule instance records results in many segment records being erased. After the final purge in step 6, only those segment records which have become part of the longer-term information structure by being reachable from a named record or MEMORY remain (e.g. R26, R6, and R20 in Figure 4.6).

5. Alternate Decoding Algorithms

An attempt has been made here to keep the decoding rule language somewhat independent of the decoding algorithm, in much the same way that a higher-level programming language is kept independent of a particular machine. Part of the reason for this is that although it is felt that the current version of the rule language is quite satisfactory, it is felt that the current decoding algorithm could be improved upon considerably by further research. As an example of one such possibility, an alternate algorithm, similar to the present one, but which does not use rule instance records, is described here. Also there is a brief discussion of parallel versus serial algorithms.

5(a). An Algorithm Without Rule Instance Records

Figure 4.7 shows a decoding algorithm that could be used in place of the one given in Figure 4.3. The essential difference between the two is that this one does not utilize rule instance records. Rather than checking on a rule's applicability from left-to-right (i.e. starting with its first constituent), it checks from right-to-left (i.e. starting with its last constituent). Because all potential constituent segments are available at that time, there is no "waiting". At any given position during the processing of a given input string exactly the same rules would be applied with either algorithm, but possibly in a slightly different order.

Although Figure 4.5, the first decoding example, was made up to illustrate the other algorithm, it can serve equally

(0) Initialize N and I to 0.

(1) Increment N by 1. Then get the next character from the input stream and enter its segment type and position in line N of the table (e.g. an "e" is a segment of type E).

(2) Increment I by 1. Then, examine each rule which has as its _last_ constituent the same segment type as appears in line I of the table. Using the information which appears in lines 1 through I, try to apply each of these rules (possibly more than once each). Each time a rule is applied, increment N by 1, and enter into line N of the table the segment type from the right side of the rule, the positions covered by the segment, and a pointer to the newly created segment record.

(3) If I is less than N, go to step 2.

(4) If the input stream is not empty, go to step 1.

(5) Halt

Figure 4.7. An Alternate Decoding Algorithm Without Rule Instance Records

well for this one. Its format is essentially that of the "table" referred to in this algorithm, with the Seg column furnishing the line number. For this particular example the order in which the rules are applied is identical for the two algorithms.

The working of this algorithm can be seen by following the development of the table for a few lines near the end of Figure 4.5. Starting at step 3 with both N and I equal to 32, control would go to step 4 and then to step 1. There, N is incremented to 33, and the "." is obtained from position 19 in the input stream, resulting in the entry on line 33. Then at step 2, I is incremented to 33, and those rules with a period (not the same as PERIOD) as their last constituent are examined. Rule 26 is the only such rule, and it can be applied, resulting in N being incremented to 34 and an entry for PUNC being made on that line, as can be seen in the figure.

Then at step 3, I (33) is less than N (34), so control returns to step 2, where I is incremented to 34 and those rules ending with a PUNC are examined. There are three such rules, 28-30. Neither of the first two can be applied because there is not an ADJP or a NOUNP with a rightmost position of 18 appearing in the table. (The PUNC is at position 19.) There is a VERBP with a rightmost position of 18 in line 32, so maybe rule 30 can be applied. The leftmost position of the VERBP is 11, so to satisfy the rule a # must be available at position 10. Looking back through the table, there is such a segment at line 18, which means rule 30 can be applied. N is incremented,

and the new VERB segment is entered on line 35. The algorithm continues in this manner until the rest of the table shown in the figure is built.

From the above discussion it may appear that I and N never differ by more than 1, but in general this would not be true. For a given input string the difference between I and N at any given time in the step 2-3 loop corresponds to the number of complete rule instance records in existence at the corresponding time in the other algorithm.

This algorithm has not been programmed, so there is no empirical measure of its performance versus the other one. It is not intuitively obvious that one is superior to the other. Although this one does not require rule instance records, it can result in having many more segment records in existence at a given time because there are no provisions for erasing them until the very end. As was pointed out earlier, the erasing of rule instance records in the other algorithm often results in the erasing of segment records that are not being used. For example, just prior to the final purge for the example of Figure 4.5, the only segment records which would be in existence are R7, R10, R12, R17, R19, and R25. Most of the others would have been erased very soon after they were created. Of course, it may be possible to build provisions into this algorithm for recognizing when at least some of these could be erased.

5(b). Parallel Versus Serial Algorithms

Both of the decoding algorithms presented in this chapter explore all possible analyses in a parallel fashion. For a

long sentence this can result in having in existence many segment records (and rule instance records in the case of the first algorithm) that turn out to not be needed. A lot of effort is expended developing "dead-end" paths along with the correct one.

Conceptually, with this approach a decision about what something is is not made until enough information is available to make the correct decision. Often this is not until the end of the sentence is reached. For example, as was seen in Figure 4.5, when the "are" in "Men are arriving." is processed, it is treated <u>both</u> as the main verb and as an auxiliary verb. Then, when "arriving" is processed, the second of these is used to further the analysis, with the first one appearing to be a dead-end. This does not mean, however, that in general the first one (i.e. considering "are" as the main verb) can be discarded as being wrong at this point. For instance, in the sentence "They are singing nuns.", when just "They are singing" has been processed, it would appear that "are" is an auxiliary verb similarly to above. However, when the end of the sentence is reached, it turns out that "are" is the main verb,and "singing" modifies "nun".

Another basically different approach is to explore possible analyses in a <u>serial</u> fashion until the correct one is found. In this case an assumption is made about what something is immediately, and then that particular path is followed until either the end of the sentence or a dead-end is reached. If it is the latter, the analysis must be "backed

up" to a decision point, where a different assumption is made and a new path followed.

If all the right decisions are made the first time through, the serial approach can produce an analysis very quickly. However, if much backing up is required, it can be very slow, especially if the analysis of that portion of the input after the decision point has to be done "from scratch" again.

It would seem that research to develop some sort of mixed algorithm that could take advantage of the strengths of both approaches might be most appropriate. The main point being made here, however, is that the rule language developed in this research is not tied to one particular algorithm, and therefore, it should be possible to devise and test other algorithms without having to write new sets of rules each time.

One further point that should be made in this regard is that although it is possible to write sets of decoding rules which are algorithm-independent that does not mean that all sets of rules automatically have that feature. This statement is similar to, "although it is possible to write FORTRAN programs that are machine-independent, that does not mean that all FORTRAN programs are machine-independent." Some sections of the rules for the queuing problem application, for instance, were written with the current decoding routine in mind and might have to be rewritten if another algorithm were to be used. These dependencies are all pretty much tied to taking advantage of knowing the order in which rules will be applied.

6. Ambiguity

The problem of dealing with ambiguity is usually brought up in connection with computer processing of natural language. Ambiguity results from having insufficient information, so the key to dealing with it is to make more information available. With the decoding rule language being presented here, all of the computer's information holding and handling capabilities are available to be used when needed during the processing of natural language text. Most of these problems can be dealt with directly in the rule language without resorting to using the Routine feature described in Section A.4(f) of Chapter III, but if need be, that feature can be used to access routines of any complexity written in FORTRAN or some other language.

At least two types of ambiguity can be distinguished -- syntactic and semantic. Syntactic ambiguity has to do with being able to give more than one phrase structure to a string of text, and semantic ambiguity has to do with being able to give more than one "interpretation" to a string of text even though it has only one possible phrase structure. Some examples of each are discussed here.

6(a). Syntactic Ambiguity

The lowest level of syntactic ambiguity occurs when the part-of-speech of a word is not unique. This is especially true of many words which can be either a noun or a verb. Usually this particular type of ambiguity can be resolved

simply by taking neighboring words into account, i.e. by determining the phrase structure of a larger portion of the text. For example, in the sentence

They will ship the ship by ship.

the word "ship" appears once as a verb and twice as a noun. Each appearance is syntactically ambiguous by itself, but when the whole sentence is considered, the part-of-speech of each "ship" is clear.

The amount of context which must be considered varies from case to case. From the example above it may seem that the last occurrence of "ship" must be a noun because it is preceded by a preposition. However, that need not always be so. In the sentence

If he comes by ship the ship.

the "ship" after "by" is the imperative form of the verb. Lengthening that same sentence slightly, can change it again:

If he comes by ship the ship may sink.

Now the "ship" after "by" is a noun. A well-placed comma could help in each of these.

In this system a morphological rule of the sort

```
S  H  I  P  -->  NOUNS('SHIP1'),
                 VERBS('SHIP2',SUFX*)
```

would result in the string "ship" being treated as both a noun and a verb, with there being a separate named record for each of the two concepts. Each of these segments would participate in the application of different rules in the

lexology, with only one of them being included in the analysis that finally covers the whole sentence. The discussion of parallel versus serial algorithms in Subsection A.5(b) is pertinent here.

A higher level of syntactic ambiguity occurs when it is not clear what is being modified by a modifying phrase. For example, in the sentence

The ship is loaded for ten hours at a dock.

just considering the fact that "hours" is a noun, the prepositional phrase "at a dock" could be modifying "hours" as well as "loaded". To resolve this type of ambiguity requires taking some semantic information into account. In this case, if "at a dock" is considered to be a <u>location phrase</u> rather than simply a <u>prepositional phrase</u>, then it would not "make sense" for it to modify the time unit "hours", but rather it must modify "loaded" (i.e. specify the location of the action).

In this system this type of syntactic ambiguity can be handled to some extent by having lexological rules of the sort

```
NOUNPH($'STATENTY',¬LOCATION)  PREPPH($'LOCDESCR',OBJ$'STATENTY')
                                    -->  NOUNPH(LOCATION=PREPPH)
```

This rule says in effect that if a noun phrase with a stationary entity noun which does not already have a location specified is followed by a prepositional phrase with a location descriptor preposition and a stationary entity object, let this prepositional phrase furnish the location for the noun phrase. The noun phrase "ten hours" and the prepositional phrase "at

a dock" would not be put together by this rule, but something like "a dock" and "in the harbor" would be.

The above rule is intended to be illustrative only, with the exact form of rules of this sort depending very much on the particular application of the system. Although this example uses terminology of the queuing problem application, it does not appear in the set of English decoding rules for that application. As will be seen in the discussion of those rules in Section B of this chapter, a more general scheme is used for accomplishing this sort of processing.

It might appear that an easier solution to this problem is to subclassify the parts of speech. For example, instead of simply having nouns, have entity-nouns, time-unit-nouns, attribute-name-nouns, etc. But, this would require having many extra, almost-identical rules to do those things that are common to all nouns, such as forming the plural. In this system those aspects of a word or phrase which need to be considered at a particular point in the analysis can be singled out and looked at, with the others being ignored temporarily.

6(b). Semantic Ambiguity

One form of semantic ambiguity occurs when a strictly syntactic analysis has no trouble determining what is being modified by a modifying phrase, but the semantic function of the modifying phrase is not clear just from the phrase structure. For example, the relationship between the two nouns is different in each of the following (where "servicing" and

"arriving" are noun forms of those verbs):

the servicing of the ship
the arriving of the ship
the color of the ship
the deck of the ship

In the first case "ship" is the goal of the action, but in the second case it is the agent, i.e. ships arrive but they do not service (at least not in the sense of mechanical servicing like what is done in a repair shop). In the third case "color" is the name of an attribute of the ship, and in the last case "deck" is the name of a part of the ship. The first two phrases above refer to actions, the next one refers to an attribute, and the last one refers to an entity.

In this system this type of ambiguity can be treated by using rules similar in form to the example lexological rule given earlier. The exact form of rules of this sort also depends very much on the particular application. Some of these are discussed in Section B for the queuing problem application.

Another form of semantic ambiguity occurs because of what is commonly called "multiple word senses". For example, "big" may mean "large", "important", or "older". In [17] Lamb discusses how context can help to disambiguate some such words, and his suggestions could be expressed in the decoding rule language of this system. However, in the queuing problem application, and probably in most other specific task-oriented applications, this form of ambiguity does not occur very frequently because the overall context of the dialogue tends

to restrict the number of possible senses for most words to just one. Also, to accomplish the task at hand (e.g. writing a GPSS program) it may not be necessary to know exactly which sense of a word is intended. One case which is treated in the rules discussed in Section B is that of "take", i.e. "take time" versus the action "take".

Still another form of semantic ambiguity occurs when pronouns are used. For example, it is straightforward to give a syntactic structure to the sentence

It waits at it.

However, to interpret it requires having information available from one or more previous sentences (e.g. "The ship arrives at a dock."). A simple heuristic for handling pronouns is included in the decoding rules for the queuing problem application,as is discussed in Section B of this chapter.

It could be seen in all of the examples discussed here that the key to dealing with ambiguity is to have sufficient information available. The attribute-value data structure of this system makes it possible to have this information available and the decoding rule language provides facilities for utilizing it.

Saying that these problems can be dealt with is not the same as saying that they have all been solved. However, it was found that in writing the rules for the queuing problem application there were no great problems with ambiguity. It is felt that this was due partly to the restricted context

in which the dialogue takes place and partly to the capabilities of this system. It is also felt that this would hold true for most other specific task oriented applications.

7. Syntax Directed Compiling

The decoding portion of this system bears some resemblance to work which has been done in the area of syntax directed compiling. In syntax directed compiling, the specification of the syntax of a programming language, usually accompanied by other information, is given to a computer program (either directly or after having been preprocessed), and then this computer program is capable of compiling programs written in the language defined by the syntax specification. Several articles on this topic are reprinted in a book by Rosen [27], and Feldman and Gries have done a comprehensive survey of work in this area [5]. In this subsection a number of specific points of similarity to this other work are briefly discussed.

In these systems the syntax of the programming language to be compiled is usually specified by means of phrase structure rules of some sort, and a crucial feature of many of these systems is that "semantic actions" may be associated with each rule, to be executed when the rule is applied. These semantic actions correspond to creation spefications in the system being described here. The form of the semantic actions differs from system to system. In the earliest work by Irons [8] its form was extremely limited, whereas in more recent

work, such as XPL [25] and APAREL [1], it takes the form of a program written in a higher-level language such as PL/I.

Because of the relative simplicity of programming languages as compared to natural languages, most of these systems do not have a capability comparable to condition specifications. APAREL does, however. It allows a PL/I routine which returns a "true" or "false" to be associated with each constituent in a rule to help determine whether or not the rule can be applied. Others base this determination strictly on syntax, but possibly considering a small amount of context. The production language of Floyd [6] is of this sort.

All of these systems use what was termed a serial algorithm in Subsection A.5(b) of this chapter. By placing suitable restrictions on the languages to be processed, "backing up" can be avoided, however, resulting in very efficient processing. Many of them utilize a "stack" which is similar to the table of the alternate algorithm given in Figure 4.7 and illustrated by the example in Figure 4.5, but with the important difference that multiple descriptions of portions of the input string do not appear there. Each time a rule is applied, the constituent segments (other than contextual) are removed from the stack before the new segment is added. This means that when searching for a rule to apply, all of the potential constituents are next to each other at the top of the stack rather than spread out all through it.

Some work has been reported on parallel algorithms for syntax directed compiling, but it appears that none of it

has been incorporated into any systems which allow semantic actions. The first paper describing a parallel algorithm was by Irons also [9]. More recently a parallel algorithm with some similarity to the first decoding algorithm given here (Figure 4.3) was described in a paper by Earley [4]. The main point of similarity is that this algorithm constructs "states", which are similar to and serve the same purpose as the rule instance records of the decoding algorithm. The main point of dissimilarity is that his algorithm works "top-down", whereas the decoding algorithm works "bottom-up". An abstract of a paper by Lang [22] describing a "bottom-up" algorithm "similar to Earley's in its organization" appeared in the proceedings of a recent conference, however.

None of the systems for syntax directed compiling have been built specifically around an attribute-value data structure. However, in systems which use a full-fledged programming language for specifying semantic actions the user could certainly build these sorts of structures if he so desired. In a system such as XPL, the stack consists of several vectors which are conceptually side-by-side. This effectively forms a table, where each row may be thought of as a segment record as defined here, and each column holds the values of a particular attribute. In two recent papers [15,16] Knuth has begun to explore the properties of "a technique of formal definition (of programming languages), based on relations between 'attributes' associated with nonterminal symbols in a context-free grammar," which may eventually furnish a formal basis for the work being described in this report.

At least one natural language processing system has been based on syntax directed compiling. Coles used this technique to convert English language statements into well formed expressions in the predicate calculus [3].

Although the system being described in this report (NLP) is intended for natural language processing, it could certainly be used for syntax directed compiling. However, its efficiency at that task is not likely to compare well with those systems which were designed for that purpose.

B. DECODING RULES FOR THE QUEUING PROBLEM APPLICATION

In Section A of this chapter the decoding process was described in detail without regard to any particular application of this system. This included discussions of the decoding rule language and the current decoding algorithm, illustrated by a sample set of rules and examples of how they work. In this section the set of English decoding rules which has been written for the queuing problem application is described.

The English decoding rules for this application are listed in Appendix I. The declarations listed in Appendix A and the named record definitions listed in Appendix B would have to be entered into the system before these rules are entered. Input processed by these rules can be seen in the sample terminal session which appears in the Introduction. Reference to the dialogue in the figures there and to the IPD in Figure 2.8 can be helpful in following the discussions of this section.

The English decoding rules in Appendix I are grouped into the three strata morphology, lexology, and semology. The morphology puts characters together to form stems and words, the lexology puts words together to form phrases, clauses, and sentences, and the semology merges information into the IPD. (Actually, some amount of this merging is done in the lexology also.) The rules of each stratum are discussed in this section.

1. English Decoding Morphology

This morphology is similar to the one included with the sample set of rules in Figure 4.1, but it does not have a rule for each stem or part. All of that sort of processing is done by rules 603-614. Rules 603-604 form a CHARSTRING segment out of a space (#) and one or more letters, using the special segment-type LETTER described in Subsection A.3(b). As can be seen in those two rules, each time a letter is added to the string, the FORTRAN Routine LOOKUP is called. This routine searches the named records for one whose name matches the string in that portion of the input stream covered by the CHARSTRING. If it finds such a named record, it sets the SUP attribute (i.e. attribute 1) of the CHARSTRING to point to it. Otherwise, the SUP remains zero (i.e. non-existent).

Then rule 605 says that a CHARSTRING with a SUP yields a STEM segment with the same SUP and with a PS (part-of-speech) attribute from the "set" of this segment. The *1 appended to STEM in that rule designates that rules with STEM on the left

will be treated like encoding rules, as discussed in Subsection A.3(c). Rules 606-614 then result in a stem (NOUNS or VERBS) or a part (NOUNP, VERBP, ADJP, ADVP, CONJP, PREPP, or PRONP), depending on the value of the PS attribute of the STEM segment. Each of these stem or part segments gets a SUP attribute pointing to the appropriate named record, just as though there had been a separate rule for each one as in Figure 4.1. DECIMAL's and UNIT's receive some special treatment in these rules.

Rules 601-602 form a NUMBERP out of a space and one or more digits, using the special segment-type DIGIT. Rules 615-617, 637-641 and 643 take care of putting together some stems and parts which can not be completely handled by the lookup procedure because they consist of more than eight characters (and the names of named records are limited to eight characters). Rules 618-636 primarily add suffixes to noun and verb stems to form parts, similarly to some rules in Figure 4.1. Rule 642 adds an "ly" to an adjective to form an adverb when it is allowed.

As an example of the application of these morphological rules, if the string "#arriving" appeared in the input stream, first the "#a" would become a CHARSTRING by rule 603. Then rule 604 would be applied four times, resulting in a CHARSTRING covering "#arriv". When the Routine LOOKUP is executed, it would put a pointer to the named record 'ARRIV' in the SUP of the CHARSTRING. Then rule 605 would be applied, resulting in a STEM with a SUP of 'ARRIV', and a PS of 'VERBS', the

For any number greater than 10, the first time it appears in a problem description a 'UNIT' record with an appropriate NUM would be created for it and added to the 'UNITLIST' in rule 723. It is also worth noting that if the string "#eight" appeared in the input stream, it too would be described by an ADJPH with a SUP of 'UNIT', a NUM of 8 and an ENTY of 'EIGHT', just as the string "#8" would be. In this case, however, the rules that would be applied to produce that are 603-605, 610, 646, and 682.

Rules 655-662 make a NOUNPH out of a NOUN. If the noun is the name of a mobile entity (i.e. if the SUP of the NOUN segment is in the set 'MOBENTY'), a search of the mobile entity list is set up by yielding a NOUN2 with appropriate attributes. The searching is done in rules 660-662 in a manner similar to that described above for SRCHREC. Names of stationary entities and miscellaneous entities are treated similarly, but using different lists. Other nouns, such as attribute names (e.g. "time") and unit names (e.g. "minutes"), become noun phrases directly in rule 659.

The purpose of list searching is to set the value of the ENTY attribute of a segment record. The ENTY attribute of a segment record points to that IPD record which is referred to by the segment of text described by the segment record. For example, whenever the string "#cars" appeared in the input for the sample problem in the Introduction, it would have been described by a NOUNPH segment which had an ENTY attribute

described by a NUMBER segment record with a SUP of 'UNIT' and a NUM of 8.

Then in rule 654 a NUMBER becomes a SRCHREC. This is worthy of attention at this point because there are several rules like this in the lexology. What it amounts to is what might be termed "using a rule subroutine." The SRCHREC segment produced here is processed by rules 723-734. This group of rules searches a specified list of records to find a record like the one it has (i.e. the SRCHREC record). If one cannot be found, one is created and added to the list. In either case the ENTY attribute of the original record is set to point to the record on the list. Then the SRCHREC becomes another type of segment according to its TYPE attribute.

For example, in rule 654 a NUMBER segment with a SUP of 'UNIT' and a NUM of 8 would yield a SRCHREC segment with the same SUP and NUM and also with the attributes LIST equalling 'UNITLIST', LC equalling 10 and TYPE equalling 'ADJPH'. The named record 'UNITLIST' can be seen on the last page of Appendix B to consist of pointers to the named records 'ONE', 'TWO', etc., each of which has a SUP of 'UNIT' and an appropriate NUM. This particular SRCHREC segment would loop through rules 723-725 eight times, and then fall through to rule 729 where it would become a SRCHREC1. This in turn would become an ADJPH in rule 731. In addition to the original attributes, the ADJPH would have an ENTY attribute pointing to the named record 'EIGHT', set in rule 724.

For any number greater than 10, the first time it appears in a problem description a 'UNIT' record with an appropriate NUM would be created for it and added to the 'UNITLIST' in rule 723. It is also worth noting that if the string "#eight" appeared in the input stream, it too would be described by an ADJPH with a SUP of 'UNIT', a NUM of 8 and an ENTY of 'EIGHT', just as the string "#8" would be. In this case, however, the rules that would be applied to produce that are 603-605, 610, 646, and 682.

Rules 655-662 make a NOUNPH out of a NOUN. If the noun is the name of a mobile entity (i.e. if the SUP of the NOUN segment is in the set 'MOBENTY'), a search of the mobile entity list is set up by yielding a NOUN2 with appropriate attributes. The searching is done in rules 660-662 in a manner similar to that described above for SRCHREC. Names of stationary entities and miscellaneous entities are treated similarly, but using different lists. Other nouns, such as attribute names (e.g. "time") and unit names (e.g. "minutes"), become noun phrases directly in rule 659.

The purpose of list searching is to set the value of the ENTY attribute of a segment record. <u>The ENTY attribute of a segment record points to that IPD record which is referred to by the segment of text described by the segment record</u>. For example, whenever the string "#cars" appeared in the input for the sample problem in the Introduction, it would have been described by a NOUNPH segment which had an ENTY attribute

pointing to REC 22 in the IPD of Figure 2.8. The use of ENTY will be seen throughout the rest of the rules.

Rules 663-665 treat personal pronouns. It can be seen there that a PRON becomes a NOUNPH with an ENTY attribute pointing to the IPD record for either the last mobile entity or the last stationary entity mentioned in the dialogue. Often two NOUNPH's are set up, one for each. Usually, because of the context, one of these will not be able to combine with another segment to become part of the final analysis. The attributes LASTME and LASTSE of MEMORY used here get their values from creation specifications in rules 656, 657, and 660. Rules 666-696 process other NOUNPH's, and rules 697-709 treat ADV's and ADJPH's.

Rule 711 puts a PREP and a NOUNPH together to form a PREPPH without checking any conditions on either constituent. The PREPPH has a SUP attribute pointing to the named record for the preposition and an OBJ attribute pointing to the segment record associated with the NOUNPH. Then in rules 712-716 the PREPPH may become an ADVIAL which is a copy of the OBJ and which has an ATTRIB attribute. For example, rule 713 would result in an ADVIAL with a SUP of 'MINUTE', a NUM of 10, and an ATTRIB of 'DURATION' to describe the string "ten minutes". The ATTRIB attribute says which attribute of an action VERBPH this ADVIAL segment could be the value of. This record would also have an ENTY attribute pointing to a similar record in the IPD.

Rule 735 furnishes an illustration of an interesting use of a contextual constituent to cause a stationary entity noun

phrase appearing immediately after a form of either the verb "enter" or the verb "leave" to be treated as if it were a prepositional phrase beginning with "at". This is done so that it can serve as the location of the action.

In rule 738 a VERB segment for any form of "take" is duplicated, with the copy having a SUP of 'TAKTIME'. Only one of these will end up as part of the final analysis. In rule 739, any form of "take" followed immediately by "place" results in a VERBPH with a SUP of 'OCCUR'. This is an example of a "lexemic sign pattern" rule, in the terminology of stratificational linguistics.

In rules 740-744 an action VERB becomes a VERBPH with a list of pointers to IPD records which have the same SUP because this phrase may be referring to any of them. Later, as attributes such as AGENT, GOAL, and LOCATION are added to the VERBPH segment, this list is pruned, so that at any time it contains pointers to just those records in the IPD which the segment may be referring to. Rules 745-793 put segments together to form bigger VERBPH's. These rules are basically like rules 34-38 in Figure 4.1 except that they handle more complex phrases, including modals, interrogatives, negatives and "be".

Rules 794-810 treat some punctuation marks and conjunctions, primarily to establish the classes DELIMB and DELIMC. These are then used as contextual constituents in rules 811-815 to determine when a VERBPH becomes a clause. Depending upon its main verb, it becomes either an attribute clause or an action clause.

Rules 816-825 process specific types of attribute clauses to yield an ATRCL of a standard form. The three main attributes of an ATRCL are ATTRIB, ENTY, and VAL. Rules 826-833 process an action clause to add attributes to the corresponding action record in the IPD. These rules yield an ACTCL. Rules 834-842 put a subordinate conjunction with an appropriate ACTCL or ATRCL to form a SUBCL. A SUBCL has an ATTRIB such as 'CONDITN' or 'SUCC' to specify what attribute of the SENT segment it can be the value of.

In rule 843 a SENT segment begins "out of nothing" in the left context of a PUNCA, which is a period, semicolon, or question mark. Then, in rules 844-846 it picks up coordinate and subordinate clauses. The former become the values of attributes 11, 12, 13, etc., and the latter become the values of attributes such as CONDITN and SUCC, depending on the value of their ATTRIB attributes. Rules 847 and 848 pick up and ignore commas and the word "and".

In rule 849 a KWSENT (key-word sentence) begins "out of nothing". Then in rule 850 each word in the sentence is added to it. If the word was found in the named records, the KWORD segment would have a SUP which it got in rule 652. When it is added to the KWSENT in rule 850, it is considered to be an attribute and given a value corresponding to the position of the word in the sentence. For example, if "program" were the third word in a sentence, the KWSENT segment would have a PROGRAM attribute with the value 3. How this is used will be seen later.

3. <u>English Decoding Semology</u>

The first three rules here (851-853) process a phrase type reply to a question asked by the encoding rules discussed in Section B.4 of Chapter III. If the reply is acceptable, ATTRIB and CENTY of MEMORY, which would have been set by those encoding rules, are used to store a pointer in the IPD to the IPD record pointed to by the ENTY attribute of the REPLY.

Rules 854-856 yield either an ACTSENT, an ATRSENT, or a QUESTION, according to the type of sentence. Rules 857-861 process the various kinds of attribute sentences (ATRSENT). Rule 859 is the one most often used, and it sets an attribute value in the IPD according to the values of the ATTRIB, ENTY, and VAL attributes of the ATRSENT segment. Rules 862-889 process the various kinds of action sentences (ACTSENT), primarily setting values of SUCCessor, PREDecessor, and CONDITN attributes of action records in the IPD. They also handle the building of successor descriptor records.

When the end of a sentence is reached without the formation of an ACTSENT, ATRSENT, or QUESTION segment, rule 890 would be applied to yield a KWDSENT (key-word sentence). This segment is then processed by rules 891-899. If a "key word" is recognized, the appropriate action is taken. For example, if the word "program" appeared in the sentence, the KWDSENT segment would have a PROGRAM attribute, as explained earlier. Rule 894 would be applied to a KWDSENT that has either of the attributes GPSS or PROGRAM, to yield a segment of the special type ENCODING,

with ETYPE(MEM) set to 'GPSSPROG'. The further processing of this segment is discussed in detail in Section B.6 of Chapter III.

Rule 901 results in an ENCODING for a QUESTION. This would go through the linkage rules of Appendix H to get to the question-answering rules of Appendix G. Rule 902 gets back to the question-asking rules of Appendix F after a response to a question asked by the system has been processed.

C. SUMMARY AND DISCUSSION

The first section of this chapter described the decoding process without regard to any particular application of this system. This included discussions of the decoding algorithm and the decoding rule language. It was seen that decoding rules are basically the same as encoding rules, but with the sides reversed. The purpose of a set of decoding rules is to convert information in the form of text in some language into equivalent information in the form of attribute-value semantic structures.

The second section of this chapter described a set of English decoding rules which have been written for the queuing problem application. These rules are separated into three strata - morphology, lexology, and semology - each of which was described separately.

The decoding algorithm has been described as being "bottom-up" and "parallel". It develops all possible analyses for each substring in the input stream in a systematic fashion.

What is especially significant is that these analyses are semantic, rather than syntactic. For example, in the queuing problem application, the segment record developed to describe an action sentence has the same form as an action record in the IPD.

The use of the ENTY attribute in the rules described in Section B of this chapter is worth noting. This attribute is associated with a segment record to specify which record in the IPD is referred to by that segment of text. In most rules where "semantic restrictions" are applied via condition specifications, the ENTY attribute is used to access the information needed to perform the test. This means, for example, that as long as a NOUNPH has an ENTY attribute pointing to the IPD record it refers to, it does not matter whether that NOUNPH came about from a NOUN or a PRONoun.

The philosophy underlying this use of ENTY is that words and phrases are "linguistic entities", and linguistic entities have syntactic features. Records in the IPD represent "real entities", and real entities have semantic features. Therefore, semantic restrictions should be applied to IPD records and not to segment records. As stated before, the ENTY attribute relates a segment of text to a record in the IPD.

Some idea of the subset of English which the decoding rules described here can handle may be obtained from the discussion and examples in Sections A.1, A.4, and A.7 of the Introduction. Although this subset is non-trivial, it is still rather small, when the "entire" language is considered.

One part which is handled fairly thoroughly, however, is the auxiliary verb system, including modals, interrogatives, and negation. To expand the capabilities of this system significantly, many more rules will have to be written.

V. CONCLUDING DISCUSSION

The specific objective set forth in the Introduction has been met. A system to enable a man to solve queuing problems through natural language dialogue with a computer has been designed and implemented. Nothing resembling this has been reported in the literature before. The system is intended eventually to make it possible for an analyst to perform a wide range of simulations on the computer without having to know a computer language. Thus, it may be considered to be an improvement on the existing methodology of computer simulation.

The general objective set forth in the Introduction has also been met. The techniques which were developed and used to implement this system should be of general applicability to a wide variety of natural language processing tasks. The rule language for specifying decoding and encoding processes is different from anything which has appeared in the literature. Although, in the last few years there have been reports of similar ideas. The language has considerable expressive power, and was quite convenient to use for specifying the processing in the queuing problem system. Thus, it may be considered to be an improvement on the existing techniques of computational linguistics.

This chapter begins with a discussion of the relationship between the work done here and stratificational linguistics. Then there is a section comparing the system developed in this

research to other natural language processing systems developed in the last few years. Finally, directions for future research are suggested.

A. RELATIONSHIP TO STRATIFICATIONAL LINGUISTICS

It was stated earlier that this work (NLP) has been done within the framework of stratificational linguistics (SL). In this section a more detailed discussion of its relationship to this theory is presented.

1. Graphic Notation

In stratificational linguistics, language is considered to be a system of relationships. It is convenient to display these relationships graphically with a network consisting of nodes connected by lines. There are seven basic nodes used in an SL network [23]:

The first four of these are AND nodes, and the last three are OR nodes. Those with more than one line at the bottom are "downward" nodes, and the others are "upward". When the lines on the end with more than one line come together at the same point, the node is "unordered". Otherwise, it is "ordered". The ordering in an AND is an ordering of the occurrence of elements in a text. The ordering in an OR has to do with priority of choices. An unordered AND implies simultaneity.

Figure 5.1 contains the same decoding and encoding rules that were used for illustration in the overview given in Section B of the Introduction. The rules are numbered in pairs because they are counterpart rules as discussed in Section A.1(e) of Chapter IV. Figure 5.2 shows an SL network drawn directly from this set of rules. It should be noted that this network describes this small portion of the English language in a more cumbersome fashion than would a network drawn "from scratch".

That part of the network corresponding to each of the five rules is circled, with the rule numbers appearing in the circles. Rules which are counterparts of one another have the characteristic that they have the same network representation. It can be seen that in the network the phrase structure side of a rule (i.e. left for a decoding rule, right for an encoding rule) is represented by a downward-ordered-AND, where each constituent is represented by a downward-unordered-AND putting together the segment-type name and the information in parentheses. The other side of the rule is represented by an upward-unordered-AND similarly putting together the segment-type name and the information in parentheses. This general form is shown in Figure 5.3.

When a rule is put into the general form shown in the figure, there may be some nodes that have only one line on the end which should have more than one line. In this case the node is not needed. Several examples of this can be seen in Figure 5.2, with the most severe example being that portion

Decoding Rules

```
1   #  A  R  R  I  V  -->  VERBSTEM('ARRIV',E,ES,ED,ING)

2   VERBSTEM(ING)  I  N  G  -->  VERB(SUP(VERBSTEM),PRESPART)

3   #  I  S  -->  VERB('BE',PRP3SG)

4   VERB  -->  VERBPHRASE(%VERB)

5   VERB('BE')  VERBPHRASE(PRESPART)  -->
                                  VERBPHRASE(FORM=FORM(VERB),PROG)
```

Encoding Rules

```
5   VERBPHRASE(PROG)  -->  VERB('BE',FORM=FORM(VERBPHRASE))
                           VERBPHRASE(-PROG,-FORM,PRESPART)

4   VERBPHRASE  -->  VERB(%VERBPHRASE)

3   VERB('BE',PRP3SG)  -->  #  I  S

2   VERB(PRESPART)  -->  VERBSTEM(SUP(VERB))  I  N  G

1   VERBSTEM('ARRIV')  -->  #  A  R  R  I  V
```

Figure 5.1. Sample Decoding and Encoding Rules

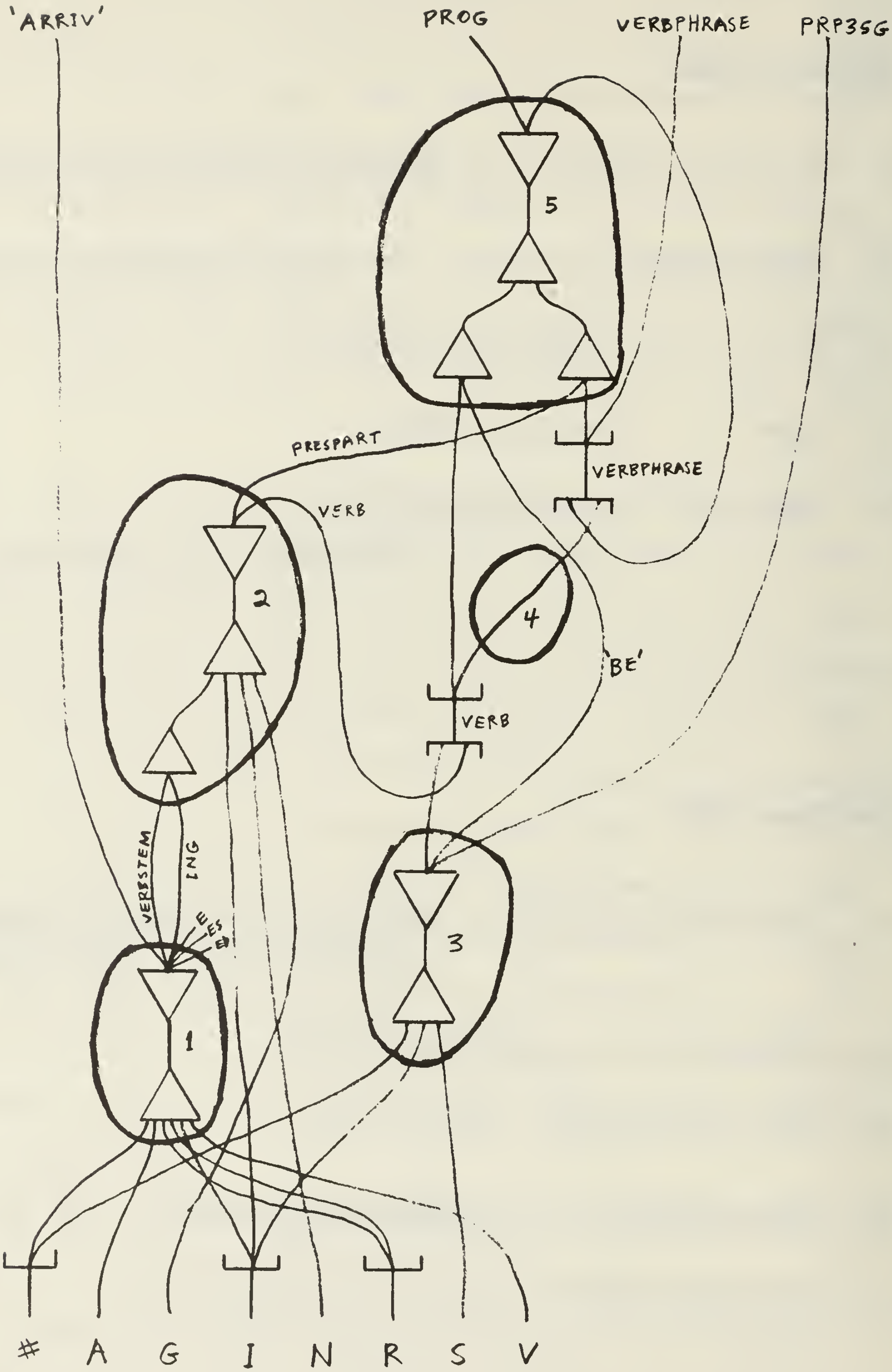


Figure 5.2. A Network Drawn from the Sample Rules

a(b,c) d(e,f) g(h,i) $\rightleftarrows$ j(k,l)

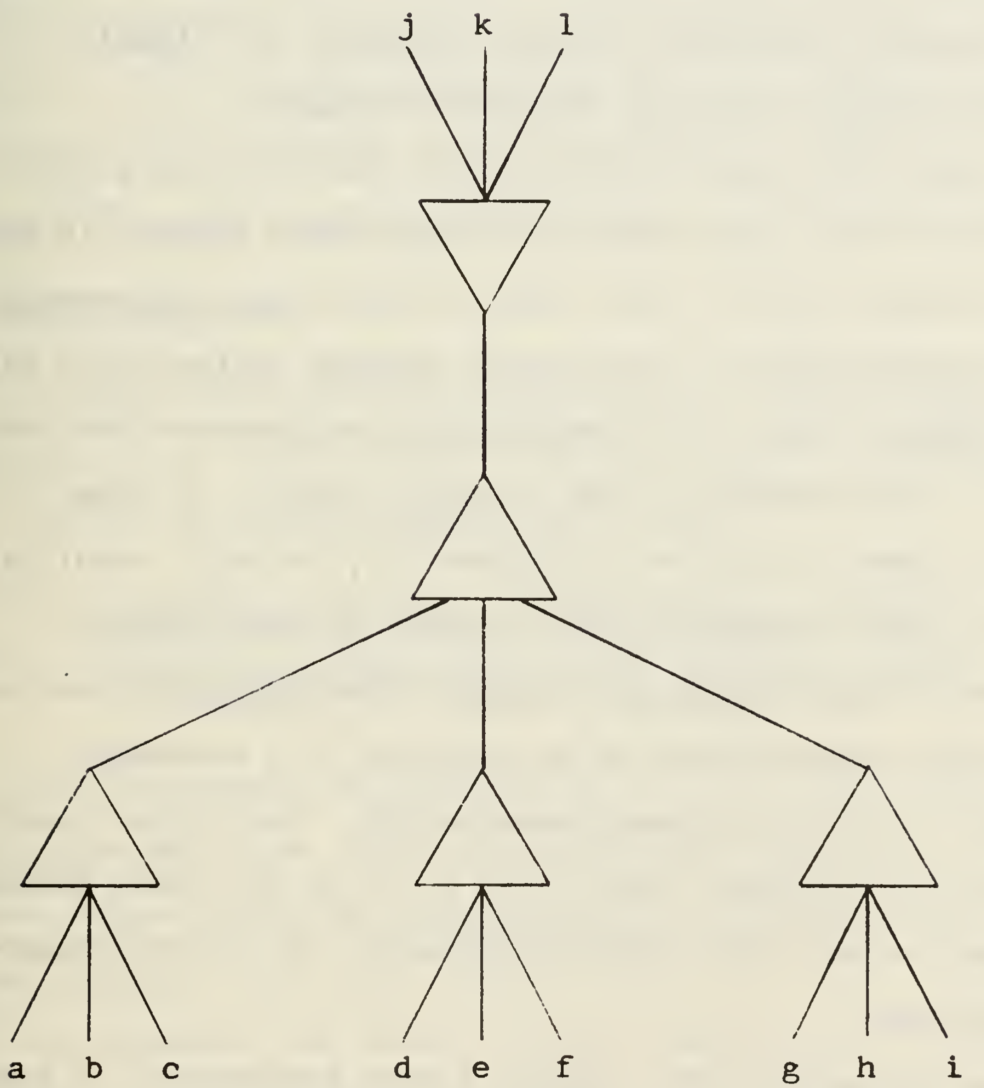


Figure 5.3. General Network Form of a Rule

of the network associated with rule 4, in which no nodes are needed.

The lines in the network are labelled to correspond to the names used in the rules. The labels on the lines internal to the network are just for convenience; the ones on the periphery may be considered to be the inputs and outputs. In an SL network, decoding involves movement of "signals" upward and encoding involves movement downward.

The use of OR nodes can be seen in this network also. An upward-OR occurs when there is a name which appears in the phrase structure side of more than one rule, and a downward-OR occurs when there is a name which appears on the other side of more than one rule. The ordering in the downward-ORs corresponds to the ordering of the encoding rules. If other rules were added to the set of Figure 5.1, it would result in additional lines to some of the OR nodes in this network. Also, some OR nodes might be inserted where there are none now.

In this network there is no instance of a downward-unordered-OR or of any upward-ordered-AND. The latter could never occur in a network drawn from a set of NLP rules because these rules do not allow multiple elements on the non-phrase-structure side.

It may be noticed that there is some information in the rules of Figure 5.1 which does not appear in the network of Figure 5.2, such as %VERB, FORM=FORM(VERB), and -PROG. If an NLP segment record were to be thought of as representing a group of signals traveling through the network, then this sort

of information is tied in with the timing of these signals to ensure that a particular group of them that are needed simultaneously at some point travel together. Timing of this sort is currently not accounted for in SL networks.

Drawing the network for this particular set of rules was fairly straightforward. However, for a larger set of rules where all of the capabilities of the NLP rule language are utilized, it is likely to be more difficult. This is an area where further study could be beneficial to both NLP and SL.

2. Basic Principles

Lockwood lists six basic principles of stratificational linguistics [23, pp 287-288]. Each of these is given here with a brief discussion relating it to NLP.

> "1. Language is a code relating concepts to articulation and audition. It is traversed in one direction in the encoding of messages, and in the opposite direction in decoding."

In NLP concepts are represented by bits in the computer's memory arranged in an attribute-value data structure. Articulation and audition are simulated by a typewriter connected to the computer. The sets of rules form the code. In NLP there are actually two codes, one for encoding and one for decoding. It has been shown, however, that some parts of each one may contain the same basic information (counterpart rules). While a single code would be ideal, it is yet to be demonstrated that one SL code of any substantial size will support both encoding and decoding.

"2. Linguistic structure consists entirely of relationships, which connect to objects only at the outer periphery of the structure."

The network of Figure 5.2 shows that NLP rules specify relationships of the sort considered here, and that they connect to objects only at the periphery. Names used in these rules are completely arbitrary.

"3. The most fundamental relationships are ANDs and ORs, which may be ordered or unordered and have either an upward or a downward orientation."

The way these relationships are present in a set of rules is also shown in Figure 5.2. What is especially significant is that condition and creation specifications are essentially <u>un</u>ordered ANDs, both upward and downward.

"4. Linguistic structure is organized into a number of stratal systems. In the period since 1966 the number posited has varied from four to six. The possibility of typological variation among languages of the world as to the number of stratal systems has been allowed for."

The rules for the queuing problem application have been grouped into three strata. These are three of the four in the four stratum model. The one missing has to do with speech, which is of no concern in this system. NLP allows any number of strata.

"5. Each stratal system contains a tactic pattern, defining the arrangement of its elements or -emes, and a realizational portion, broken into sign and alternation patterns."

In the rules that have been written, no attempt has been made to keep the patterns separate, but examples from all

three can be found. In many cases, a single rule overlaps patterns. For instance, in the English decoding lexology of Figure 4.1, rule 38, which puts together a subject and a predicate, checking for number agreement, is a good example of a tactic pattern type. In the morphology of that same figure, rules 1-11, which build words, furnish sign pattern examples. Rules 24 and 25 for "are" and "is" are primarily alternation pattern types, but they also include sign patterns.

> "6. The tactics provides the environments for alternate realizations within each stratal system, though the precise means of implementing this provision have varied."

In an NLP rule the information outside the parentheses (i.e. the non-terminal segment-type names and their arrangement) corresponds roughly to the tactics, and the information inside the parentheses corresponds roughly to the realization portion. An example of the tactics providing the environment for alternate realizations can be seen in the English encoding rules in Appendix C, where the agent of an action may be realized as simply a noun phrase or as a prepositional phrase beginning with "of", depending upon whether the overall action is being realized as a verb phrase (e.g. "ships arrive") or as a noun phrase (e.g. "arrival of ships").

3. Contribution to Stratificational Linguistics

The research being discussed in this report may be able to contribute to the theory of stratificational linguistics. It has been seen here that the system developed is consistent with that theory. It has also been seen that this system

"works", i.e. it is capable of decoding and encoding. The only working model that has been reported so far using the usual SL notation is Reich's Relational Network Simulator [26]. In his general network processor, which runs on an IBM 7094 computer, the operation of the nodes is specified by defining each as a finite-state machine. The only example given in his paper is for encoding with a grammar for that portion of English syntax and morphology concerned with auxiliary verbs.

By studying in more detail the relationship between the NLP rule language and the SL network notation, it may be possible to devise variants of the NLP decoding and encoding algorithms which would work for the networks. It seems especially likely at this time that some means of synchronizing the signals traveling in the network will need to be devised. In NLP, segment records accomplish essentially that. It may turn out that the NLP rule language is simply a convenient notation for describing an SL network.

B. COMPARISON TO OTHER NATURAL LANGUAGE PROCESSING SYSTEMS

The work done here falls into that area of artificial intelligence called "natural language question-answering systems." Two comprehensive surveys of research in this area have been done by Simmons [34,36]. Since this project was begun, a number of reports of other work with similar natural language processing goals have appeared in the literature. Although the work on NLP was done independently, the system developed here has certain features in common with each of

these others. In this section the similarities and differences between NLP and the others are discussed.

1. <u>REL (Thompson)</u>

In a 1969 paper [40] Thompson, et al, defined REL (Rapidly Extensible Language) as "an integrated information system designed to facilitate conversational access to a computer." In the paper both the general REL system and a specific application language called REL English are discussed. REL English is oriented toward interpreting and answering questions about a data base.

The data base is organized in "ring structures". Such structures are somewhat similar to the attribute-value structures used in NLP, but there is more inter-connectivity among the elements. Numerical information about an element (e.g. its population) is stored exactly as an attribute in NLP is, but relations between elements (e.g. location) require going through an intermediate element for the relation. The data base appears to hold primarily specific information roughly of the sort held in an IPD, but on a much larger scale. It probably also holds concept information. Nothing is said about how information about words is held, other than that they are expressed to the system implicitly through the grammar rules.

The REL language processor is a "syntax-directed interpreter." The grammar is specified by rules which are basically phrase structure rules. However, on the non-phrase-structure side a "syntax completion routine" (which may simply be

a syntactic category name) and a "semantic routine" may be specified. When a rule is applied, the syntax completion routine is executed and the semantic routine may or may not be. Those routines together are intended to serve the same roles that condition and creation specifications do together in NLP. The paper does not say what programming language these routines must be written in. The parsing algorithm appears to be bottom-up and parallel.

In REL English the analysis of a sentence is done somewhat similarly to the way it is done by the decoding rules of NLPQ. During processing of a clause a "verb table" is set up. This table serves essentially the same role as a VERBPH segment record and has entries such as subject, object, and times. The phrase structure of the clause is developed in almost the same order as is done by the NLPQ rules, also. "Features" in REL are similar to indicators in NLP and are used extensively in REL English for similar purposes as indicators in NLPQ, e.g. NEG, PASSIVE, and even PRM.

The paper states that sentences in REL English may be questions which interrogate the data base or commands which modify it, but no details are given about the latter. The system can answer questions which require much more processing than those handled by the current NLPQ rules, but the answers are not in the form of English sentences.

REL appears to provide no facilities comparable to the encoding portion of NLP.

2. Conceptual Dependency (Schank)

Conceptual dependency is a linguistic theory being developed by Schank and his coworkers at Stanford. Since 1968 a number of papers and reports describing this work have appeared. The cornerstone of this theory has been that conceptual analysis is more important than syntactic analysis in the understanding of text (by man or machine) [28]. This theory has been influenced by stratificational linguistics.

A 1969 paper by Schank and Tesler [29] describes a "conceptual parser." The underlying philosophy of this parser is in basic agreement with that stated here for NLP, i.e. a string of text is more usefully described by a semantic structure than by a syntactic structure. Therefore, during the processing of a string, a semantic representation of it is built to show the relationships between the concepts involved (not between the words as a syntactic representation would). This representation is in the form of "conceptual dependency links" rather than attribute-value pairs, but displays essentially the same information as a final SENTence segment in NLPQ would, such as actor and object (i.e. AGENT and GOAL).

The parser described in this paper is guided by "realization rules" which tell what sort of conceptual structure to build when certain kinds of sequences of elements are found in the input. These differ from phrase structure rules in that the elements are conceptual classes rather than syntactic classes and the elements need not be adjacent in the string for the rule to be applied. In the data base available to the parser,

there is a "conceptual semantics" associated with each concept to specify the permissable relationships between it and other concepts. (This is similar to the semotactics of a stratificational grammar.) When a realization rule is applied, the structure built is aborted if it violates the conceptual semantics. In the NLPQ decoding rules this sort of thing is accomplished by condition specifications, but on a much smaller scale than is done here. From the paper it appears that the parsing algorithm is theoretically parallel but that the implementation is serial. The application mentioned is an on-line dialogue program for psychiatric interviewing.

A 1970 report by Schank, Tesler, and Weber [31] presents revisions both to the theory and to the parser described earlier. The changes to the parser move it further in the direction away from a phrase-structure parser, with the basic assumption now being that conceptual rules rather than realization rules should be responsible for the bulk of the parsing. Heuristics play an important part in this new procedure. Syntactic information is not ignored, but it simply does not have the dominant position that it has in NLP and other systems.

Most recently, the work on Conceptual Dependency Theory seems to be directed primarily at developing better conceptual representations by discovering "primitive concepts" [32, 33]. In current linguistic terminology this means developing deeper "deep-structures". With the present NLPQ rules the sentences "John sold a car to Mary" and "Mary bought a car from John" would be treated as two different actions. In a deeper

representation they would be identical. NLP could certainly support a deeper representation, but for the initial application it has not been needed.

Although the generation of text from a conceptual representation has been discussed [30], it is not clear what sort of computer implementation, if any, has been done.

3. A Procedure Model (Winograd)

In 1971 Winograd reported on a system for carrying on a dialogue in English with a simple hand-eye robot [43]. The robot can be given instructions to manipulate toy objects (primarily blocks of various sizes, shapes, and colors), it can be interrogated about the scene, and it can be given information it will use in deduction. Stated simply, Winograd's basic premise was that when doing natural language processing on a computer, the computer should be used as a computer, i.e. the system should not be restricted by some non-computer theory of language. Consequently, most of the information in the system is in the form of programs rather than data, especially the grammar. Even some dictionary definitions take this form.

Those data structures that do exist are LISP list structures. Information about the current scene is maintained in a "logical format", consisting of objects and assertions, rather than in an attribute-value format.

The parsing of an input string results in the building of a syntactic structure and also a semantic structure. The syntactic structure is basically a phrase structure tree, but with

the important exception that each node has a number of features attached to it. This is part of the information normally contained in an NLP segment record. It is different, however, in that a segment record does not keep the constituent structure. (Discussion relevant to this is included in Section A.2(c) of Chapter IV.) At certain points during the building of the syntactic structure a semantic structure in the form of a program in the PLANNER language is built also and associated with it. The parser is top-down and serial, although there is some parallel processing with semantics. There is no automatic backing up, but rather the grammar program must handle it explicitly.

The system responds in English, but usually not in complete sentences. This encoding appears to be done in an ad hoc fashion, without utilizing a grammar.

4. Augmented Transition Network Grammars (Woods)

A "recursive transition network" is a directed graph with labeled states and arcs, a distinguished state called the start state, and a distinguished set of states called final states. Every such network has an equivalent context-free phrase structure grammar. In a 1970 paper [44] Woods suggested adding to each arc of the recursive transition network an arbitrary condition which must be satisfied in order for the arc to be followed, and a set of structure building actions to be executed if the arc is followed. This version he calls an "augmented transition network" (ATN).

The basic similarity between this and the idea behind the NLP rule language should be apparent. The details are different,

however. The formal ATN model provides sets of named "registers" in which to hold information during processing. A set of registers corresponds closely to a segment record in NLP, in that it is used to hold information about a portion of the input string. Woods' implementation of the ATN model is in LISP, so conditions and actions can be arbitrarily complex LISP functions. The parsing algorithm is serial.

An augmented transition network grammar has been used for at least one large-scale application, a fact retrieval system that answers a geologist's questions posed in English about the reported chemical analyses of the lunar rock samples brought back from the Apollo missions [45]. In this system an ATN grammar is used to produce a syntactic structure from an input question. Then a semantic interpretation is done on this structure using pattern-action rules to produce a formal query in a language which is a generalization of the predicate calculus. Finally, an answer is produced, but not in the form of an English sentence. The listings of the ATN grammar and the semantic rules for this application comprise more than 100 pages in the appendices.

5. Semantic Networks (Simmons)

This year Simmons has reported on a system for natural language processing that comes very close to NLP in all three areas-information structure, decoding, and encoding [38]. The information structure is in the form of attribute-value records which are quite similar in content to those of NLPQ. Information

about words and concepts is entered initially and is used to guide the parsing. During parsing, records to represent specific actions and entities appearing in the text are built. These semantic structures are described in more detail in an earlier report [37].

Parsing in this system is done using an augmented transition network grammar. Rather than building a syntactic structure as Woods does, however, Simmons builds an attribute-value semantic structure of the same sort as built by the NLPQ decoding rules.

Generating of text in this system is also done with an augmented transition network grammar. Attribute-value semantic structures are converted into strings of text in a manner which is similar to that of a portion of the NLPQ English encoding lexology. A recent paper [39] describes this processing in more detail, including the production of multi-sentence discourse.

6. <u>A Grammar-Rule Language (Kaplan)</u>

In a 1970 RAND report [10] Kaplan gives "a formal description of the MIND Grammar-Rule Language, a notation for the syntactic rules of natural languages." What makes it of interest here is that "the objects referred to in the rules are characterized by unrestricted lists of features each of which takes the form of an attribute and an associated value." However, the report is not very clear and there has been no indication in the literature that this grammar-rule language

has been implemented, so nothing more will be said about it here.

7. <u>Discussion of General Systems</u>

What all of the systems discussed in this section have in common is an underlying philosophy that during the processing of natural language text, data structures showing more than just the arrangement of items in the text are needed. In other words, "deep structures" rather than "surface structures" should be built while parsing. As has been seen throughout this report, this is exactly the philosophy of NLP.

The depth of these structures and their manner of representation in the computer varies from system to system. Schank's and Winograd's would be the deepest, Woods' and Kaplan's would be the shallowest, and Simmons' and Thompson's would be somewhere in between, as would NLP. The attribute-value manner of representation as used by NLP is also used explicitly by Simmons and Kaplan. The others use different structures, some of which are clearly equivalent and some of which are not.

The basic parsing scheme used by all of these systems, including NLP, is what might be called a "condition-action" scheme, which says "if such-and-such a condition exists, perform a particular action." For a simple phrase structure parser the only condition is on the arrangement of symbols, both terminal and non-terminal, in an input string, and the only action is to replace one or more symbols with a new symbol.

However, in all of the systems discussed here, including NLP, both the conditions and the actions are limited only by what can be stated with a computer program.

The manner in which conditions and actions are specified varies from system to system. Conditions on the arrangement of symbols are given by phrase structure rules in Thompson's, Kaplan's, and NLP, by realization rules in Schank's, by a transition network in Woods' and Simmons', and by a program in Winograd's. In all of the systems, procedures written in some programming language, typically LISP, can be used to express conditions on other than the arrangement of symbols. For the most part, actions similarly are expressed by procedures. Schank, however, does show the structure to be built in his realization rules.

NLP is the only one of these systems implemented for which a particular type of data structure, other than strings, is referred to directly from the rules for both conditions and actions. Attribute-value records are used, and facilities for conveniently interrogating and manipulating them during parsing are available. The registers of augmented transition networks serve a somewhat similar purpose, but the facilities provided for manipulating them appear to be more primitive. An especially nice feature of the NLP rule language is that all processing is specified in the rules, rather than in a separate set of procedures.

Simmons' system is the only one of those described here that generates text on a scale comparable to NLP. Even though the ATN model is used for both parsing and generating in his

system, there seems to be more of a difference between those two than there is between decoding and encoding in NLP, however.

Each of the systems described here, except possibly Thompson's, uses a separate routine to do morphological analysis. In NLP this is conveniently done in the same rule language. At the upper end, semantic processing is also conveniently expressed in the same rule language. Actually, all processing - decoding, encoding, and even setting up dictionary type information - is expressed in the same language in NLP.

8. Discussion of Specific Applications

Of the systems discussed here the only ones with which specific "serious" applications have been attempted are Woods' and Thompson's. Information retrieval systems have been described with both. Winograd has an application, but it was chosen to illustrate his techniques, rather than because there was a need for such a system. Because of the very special nature of his application, it is not clear just how generally applicable his techniques are. (However, it should be noted that it is likely that for at least the next several years this charge could be levelled against any natural language processing system developed.)

NLP has been developed with a serious application in mind, that of automating the task of simulation programming. Toward this goal several sets of encoding and decoding rules, along with named records, have been written to produce the specific system NLPQ.

Developing a specific system for a serious application helps to set the goals of the research. Rather than "picking and choosing" amongst the myriad of problems that are known to exist in natural language processing, the application dictates which ones have to be solved, and which ones are unimportant.

The NLPQ system builds what Simmons would call a semantic network (i.e. the IPD) for an entire text, not for just an isolated sentence. This immediately gets into the problem of reference, i.e. determining what object in the semantic network is being referred to by a string of text. This system can lead the dialogue by asking questions. In order to know what questions to ask, the system has to know what information it needs for the task at hand. The NLPQ system can write an entire English paragraph from the information in the semantic network, and it can also write a GPSS program, using similar techniques. These features pointed out here make NLPQ different from any other system reported in the literature to date.

C. DIRECTIONS FOR FUTURE RESEARCH

The work described in this report is considered to be a beginning rather than an end. A general system to support natural language processing (NLP) has been developed to the point that it could be used to develop an initial version of a specific system for performing simulation analyses through natural language interaction (NLPQ). Development of the general system was guided by the requirements of developing the specific

system. In this section suggestions for future work in both the general and specific systems are given.

1. <u>The General System</u>

There is work of both a theoretical and a practical nature to be done with the general system. On the theoretical side, further exploration of the relationship between NLP and stratificational linguistics could benefit both, as was stated earlier in this chapter. Similarly, the relationship between NLP and augmented transition network grammars should be explored. Other decoding algorithms should be devised and tested, as discussed in Section A.5 of Chapter IV.

As a practical matter, the rule language could be improved in a number of ways. The "condition-on-the-right" described in Section A.6(c) of Chapter III is a very handy device. Therefore, it might be a good idea to develop a more powerful conditional capability to use in creation specifications to give greater control over the order of execution of the elements. In the NLPQ rules there are a number of places where it would be convenient to be able to build subordinate records from scratch from within a creation specification, rather than having to copy and modify another record. A facility for doing this could be added to the language. Also, a capability for dealing with full-word integers and real numbers would be useful for some applications.

Probably the best way to improve NLP would be to develop other specific applications to see what features they require.

The only application other than NLPQ which has been done so far is a modified version of ELIZA [42].

2. The Queuing Problem Application

There is work of both a theoretical and a practical nature to be done with this specific system also. A number of suggested improvements have already been stated in earlier chapters. In addition, there is still a great deal of work to be done on extracting information correctly from English phrases and sentences (i.e. "understanding" them). The problem of relating strings of text to objects in the IPD has not been solved in general either.

To increase the usefulness of NLPQ, it should be expanded to cope with more of the situations that may occur in queuing problems. To do this, for each particular type of situation that is to be dealt with, it must first be determined how the relevant information for such a situation is to be stored in the IPD. Then the manner in which this information might appear in an English statement must be determined and additional decoding rules written to process it, if necessary. Most likely new words and concepts would be involved, so additional named record definitions would have to be written too. Additions may have to be made to the English encoding rules for stating this information.

The type of GPSS code required for the simulation with this new situation would have to be determined, and GPSS encoding rules written to produce this code. Also, if there

were questions that the system could ask to obtain missing information relevant to this new situation, additions would have to be made to the encoding rules for asking questions. The form of each of the additions mentioned here would be influenced somewhat by each of the others. What should be searched for in the long run are the most primitive building blocks needed for describing queuing problems.

APPENDIX A

Declarations for

Attribute,

Indicator,

and

Routine Names

```
INDICATORS:

          SUFX  1-10
              NSFX  1,      E  2,      S  3,      ES  4
              ING  5,       ED  6,     ER  7,     EN  8
              PT  9-10,       N  9,       R  10
          INCOMP  11
          TRANS  12
          VERBPHIND  13-33
          VERBPIND  13-24
            VFORM  13-18
               FINITE  13-15,          PAST  13
                  NUMB  14-15,         SING  14,            PLUR  15
               INF  16       INFIN  16,          PRESPART  17,
               PASTPART  18,           NEG  19
            INFF  20,        PASTF  21
            AUXIL  22-27
            MODAL  22-24,     FUTURE  22,   DO  23,   CAN  24
          PASSIVE  25,       PROG  26,           PERFECT  27
          TOINF  28,     NOUNAL  29,     INTERG  31,     THERE  32
          PRM  33,      QUESMK  34,     NUMBSUBJ  36-37
          MARK1  46,      MARK2  47
          REACHED  48,      CHECKED  49,      LIMITCHK  50
          POSS  13
          PRONIND  11-15,      PERS1  11,      PERS2  12
          LY  1
          EXPUSED  1,       NORMUSED  2,          DOLLARSIGN  3
          WRECOGNIZED  4,             SPARSED  5,          SUNDERSTOOD  6
          QUESTSW  7,      NOREST  8,      SUCCCHANGED  9,      QAMODE  10
          GPSSSW  11,     XVSW  12,     EXPNORM  13,     NOCOPY  14

ATTRIBUTES:

          SUP  1,  INDIC  2,        STRUC  9,      NAME  10
          IX  8, XV  9
          ARGA  11, ARGB  12, ARGC  13, ARGD  14, ARGE  15, ARGF  16,
          ARGG  17, ARGH  18

ROUTINES:

          LOOKUP  1,       CONVERT  2,     ENDOFWORD  3,
          FORMSYMBOL  4,
          MESSAGE1  6,     MESSAGE2  7,    MESSAGE3  8
          MESSAGE4  9
          GETYPE  10

:END OF FILE:
```

A P P E N D I X   B

Named Record Definitions

```
NAMED RECORDS:

          A          ('DET')
          ABSTIME    ('QUANVAL',ATTRIB='TIME')
          ABSWEIT    ('QUANVAL',ATTRIB='WEIGHT')
          ABSQNTY    ('QUANVAL')
          ABSCOLR    ('QUALVAL',ATTRIB='COLOR')
          ABSTATE    ('QUALVAL',ATTRIB='STATE')
          ACTIVITY   ('ACTION')
          ACTION     (PS='VERBS')
          AFTER      (PS='CONJ',ATTRIB='PRED')
          AGENT      ('ATTR')
          AN         ('DET')
          AND        (PS='CONJ')
          ARE        ('BE',PLUR,PS='VERB')
          ARGVAL     ('VALU')
          AROUND     ('LOCDESCR')
          ARRIV      ('EVENT',E,ES,ING,ED,ER)
          AS         (PS='ADV')
          AT         ('LOCDESCR')
          ATTR       (PS='NOUNS')
          ATTRVERB   (PS='VERBS')
          AVAILABL   ('ABSTATE',INCOMP)
          BANK       ('STATENTY')
          BARGE      ('STATENTY')
          BE         ('ATTRVERB',ING,EN)
          BECOM      ('ATTRVERB',E,ES,ING)
          BEFORE     (PS='CONJ',ATTRIB='SUCC')
          BETWEEN    (PS='PREP')
          BIG        ('RELSIZ',SYN='LARGE')
          BLACK      ('ABSCOLR')
          BLUE       ('ABSCOLR')
          BUSY       ('ABSTATE')
          BUT        (PS='CONJ')
          BY         ('LOCDESCR')
          CAN        (CAN,PS='VERB',SING,PLUR)
          CAPACITY   ('ATTR')
          CAR        ('VEHICLE')
          CARGO      ('ENTITY',ES)
          COLOR      ('ATTR')
          COMMERCE   ('TYPEVAL')
          COMMISC    ('TYPEVAL')
          CONDITN    ('ATTR')
          COND       ('VALU')
          COND1      ('COND',SMOD="SNF",FMOD="NU")
          COND2      ('COND',SMOD="SF",FMOD="U")
          CONSUMP    ('ATTR')
          CUSTOMER   ('PERSON')
          DARK       ('RELCOLR')
          DAY        ('ABSTIME',NUM=24,UNITS='HOUR')
          DECIMAL    ('ABSQNTY')
          DEPOT      ('STATENTY')
          DET        (PS='ADJ')
          DEVIATIO   (PS='NOUNS',INCOMP)
          DISTR1     ('FNCTN')
          DISTRIBU   (PS='ADJ',INCOMP)
          DO         (DO,PS='VERB',PLUR)
          DOES       (DO,PS='VERB',SING)
          DOCK       ('STATENTY')
          DURATION   ('ATTR',INTRGPH=" HOW LONG")
          EIGHT      ('UNIT',NUM=8)
          EMPDIST    ('DISTR1')
          ENTER      ('EVENT',NSFX,S,ING,ED)
          ENTITY     (PS='NOUNS')
          EQ         ('RELIND1')
          EQUAL      (PS='ADJ')
          ER         ('RELIND2')
          EST        ('RELIND2')
          EVENT      ('ACTION')
          EVERY      (PS='PREP')
          EXPON      ('STDIST',LY,INCOMP)
          FAST       ('RELSPEED')
          FEW        ('RELQNTY')
          FILLER     (PS='ADV')
          FIVE       ('UNIT',NUM=5)
          FNCTN      ('VALU')
          FOR        (PS='PREP')
          FOURTH     ('DECIMAL',NUM=250)
          FOUR       ('UNIT',NUM=4)
          FRACTNL    ('SUCDSCR1')
          FREE       ('ABSTATE')
          FROM       (PS='PREP')
          FTYP       ('SUCDSCR2')
          FULL       ('ABSTATE')
          FUNCNO     ('ARGVAL')
          GASSTA     ('STATENTY')
          GOAL       ('ATTR')
          GOTO       ('EVENT')
```

```
GRAY      ('ABSCOLR')
GREEN     ('ABSCOLR')
GREATER   (PS='ADJ')
GT        ('RELIND1')
HALF      ('DECIMAL',NUM=500)
HAPPEN    (PS='VERBS',NSFX,S,N)
HARBOR    ('STATENTY')
HAS       ('HAV',SING,PS='VERB')
HAV       ('ATTRVERB',E,ING,TRANS)
HE        ('PERS',PERS1,SING)
HEAVY     ('RELWEIT')
HELD      ('HOLD',PASTPART,PASTF,PS='VERB')
HER       ('PERS',PERS1,SING)
HIM       ('PERS',PERS1,SING)
HOLD      ('ATTRVERB',NSFX,S,ING,TRANS)
HOUR      ('ABSTIME',NUM=60,UNITS='MINUTE')
HOW       (PS='ADV')
HOWLONG   ('INTRGVAL',ATTRIB='DURATION')
HOWOFTEN  ('INTRGVAL',ATTRIB='IETM')
IDNO      ('ATTR')
IDNAME    ('ATTR')
IETM      ('ATTR',INTRGPH=" HOW OFTEN")
IF        (PS='CONJ')
IMMEDIAT  ('FILLER',PS='ADJ',LY,INCOMP)
IN        ('LOCDESCR')
INTRGVAL  ('VALU')
IS        ('BE',SING,PS='VERB')
IT        ('PERS',PERS2,SING)
JUST      ('FILLER')
LARGE     ('RELSIZ',SYN='BIG')
LAVENDER  ('ABSCOLR')
LEAV      ('EVENT',E,ES,ING,ER)
LEFT      ('LEAV',PASTPART,PASTF,PS='VERB')
LENGTH    ('ATTR')
LESS      (PS='ADJ')
LIGHT1    ('RELCOLR')
LIGHT2    ('RELWEIT')
LINE      ('ENTITY')
LINEREC   ('LINE')
LITTLE    ('RELSIZ',SYN='SMALL')
LOAD      ('ACTIVITY',NSFX,S,ING,ED,ER,TRANS)
LOCATION  ('ATTR',INTRGPH=" WHERE")
LOCAT     ('ATTRVERB',E,ES,ING,ED,ER)
LOCDESCR  (PS='PREP',ATTRIB='LOCATION')
LONG      (PS='ADV')
LT        ('RELIND1')
MAN       ('PERSON')
MANY      ('RELQNTY')
MAUVE     ('ABSCOLR')
MEAN      ('ATTR')
MEN       ('MAN',PLUR,PS='NOUN')
MINUTE    ('ABSTIME',NUM=60,UNITS='SECOND')
MISC      ('TYPEVAL',INCOMP)
MOBENTY   ('ENTITY')
NE        ('RELIND1')
NEAR      ('LOCDESCR')
NG        ('RELIND1')
NINE      ('UNIT',NUM=9)
NL        ('RELIND1')
NORMAL    ('STDIST',LY)
NOT       (PS='ADV')
NULL      (CHECKED)
OCCUR     (PS='VERBS',NSFX,S,R)
OF        (PS='PREP')
OFTEN     (PS='ADV')
ON        ('LOCDESCR')
ONE       ('UNIT',NUM=1)
ORANGE    ('ABSCOLR')
OTHER     ('DECIMAL',NUM=1000)
OTHERWIS  (PS='ADV',INCOMP)
OUNCE     ('ABSWEIT')
OWNER     ('ATTR')
PARAM1    ('PARAMNO',NUM=1)
PARAMNO   ('ARGVAL')
PERSON    ('MOBENTY')
PER       (PS='PREP')
PERCENT   ('ABSQNTY')
PERSONAL  ('TYPEVAL')
PERMISC   ('TYPEVAL')
PERS      (PS='PRON')
PIER      ('STATENTY')
PLACE     (PS='NOUNS')
PORT      ('STATENTY')
POUND     ('ABSWEIT')
PRED      ('ATTR')
PROBTIME  ('ATTR')
PROBLEM   (PS='NOUNS')
PTYP      ('SUCDSCR1')
PUMP      ('STATENTY')
PURPLE    ('ABSCOLR')
```

```
QTYP      ('SUCDSCR2')
QUANTITY  ('ATTR')
QUARTER   ('DECIMAL',NUM=250)
QUANVAL   ('VALU',PS='NOUNS')
QUALVAL   ('VALU',PS='ADJ')
RANGE     ('ATTR')
RANDMREC  ('RANDM')
RANDM     ('ARGVAL')
RATE      ('ATTR')
RED       ('ABSCOLR')
RELIND1   ('RELTV')
RELIND2   ('RELTV')
RELSPEED  ('RELVAL',ATTRIB='SPEED')
RELQNTY   ('RELVAL',ATTRIB='QUANTITY')
RELCOLR   ('RELVAL',ATTRIB='COLOR')
RELWEIT   ('RELVAL',ATTRIB='WEIGHT')
RELSIZ    ('RELVAL',ATTRIB='SIZE')
RELTV     ('VALU')
RELVAL    ('VALU',PS='ADJ')
REST      ('DECIMAL',NUM=1000)
SAME      (PS='ADJ')
SECOND    ('ABSTIME')
SERVIC    ('ACTIVITY',E,ES,ING,ED,ER,TRANS,AGORGL='GOAL')
SEVEN     ('UNIT',NUM=7)
SHE       ('PERS',PERS1,SING)
SHIP      ('MOBENTY')
SHOULD    (DO,PS='VERB',SING,PLUR)
SIMULATI  (PS='NOUNS',INCOMP)
SIMPLY    ('FILLER')
SIX       ('UNIT',NUM=6)
SIZE      ('ATTR')
SLOW      ('RELSPEED')
SMALL     ('RELSIZ',SYN='LITTLE')
SPEED     ('ATTR')
STATE     ('ATTR')
STATION   ('STATENTY')
STATENTY  ('ENTITY')
STANDARD  (PS='ADJ')
STDEV     ('ATTR')
STDIST    ('VALU',PS='ADJ')
STYP      ('SUCDSCR2')
SUCC      ('ATTR')
SUCDSCR1  ('SUCDESCR')
SUCDSCR2  ('SUCDESCR')
TABL      ('FNCTN')
TAKTIME   ('ATTRVERB',TRANS)
TAK       ('EVENT',E,ES,ING,EN,ER,TRANS)
TEN       ('UNIT',NUM=10)
THAN      (PS='CONJ')
THE       ('DET')
THESE     (PS='ADJ')
THERE     (PS='ADV')
THEN      (PS='ADV')
THEY      ('PERS',PERS1,PERS2,PLUR)
THEM      ('PERS',PERS1,PERS2,PLUR)
THIRD     ('DECIMAL',NUM=333)
THIS      (PS='ADJ')
THREE     ('UNIT',NUM=3)
TIME      ('ATTR')
TO        (PS='PREP')
TON       ('ABSWEIT')
TOOK      ('TAK',PASTPART,PASTF,PS='VERB')
TRUCK     ('VEHICLE')
TWO       ('UNIT',NUM=2)
TYPE      ('ATTR')
TYPDIST   ('DISTR1')
TYPTABL   ('TABL')
TYPEVAL   ('QUALVAL',ATTRIB='TYPE')
UNAVAIL   ('ABSTATE',INCOMP)
UNIFORM   ('STDIST',LY)
UNIT      ('ABSQNTY',PS='ADJ')
UNLOAD    ('ACTIVITY',NSFX,S,ING,ED,ER,TRANS)
UNTIL     (PS='CONJ')
VEHICLE   ('MOBENTY')
VIOLET    ('ABSCOLR')
WAIT      ('ACTIVITY',NSFX,S,ING,ED,ER)
WEIGHT    ('ATTR')
WHAT      ('INTRGVAL',PS='ADJ')
WHATREC   ('WHAT',PRM)
WHEN      (PS='CONJ',ATTRIB='PRED')
WHERE     ('INTRGVAL',ATTRIB='LOCATION',PS='ADV')
WHICH     ('INTRGVAL',PS='ADJ')
WHITE     ('ABSCOLR')
WILL      (FUTURE,PS='VERB',SING,PLUR)
WINDOW    ('STATENTY')
WITH      (PS='PREP')
YELLOW    ('ABSCOLR')
```

```
SEEDS     ('RMULT',@11=277,@12=423,@13=715,@14=121,@15=655,@16=531,
                    @17=999,@18=813)

EXPONREC  (@101=0,@102=0,@103=100,@104=104,@105=200,@106=222,
           @107=300,@108=355,@109=430,@110=509,@111=500,@112=690,
           @113=600,@114=915,@115=700,@116=1200,@117=750,@118=1390,
           @119=800,@120=1600,@121=840,@122=1830,@123=880,@124=2120,
           @125=900,@126=2300,@127=920,@128=2520,@129=940,@130=2810,
           @131=950,@132=2990,@133=960,@134=3200,@135=970,@136=3500,
           @137=980,@138=3900,@139=990,@140=4600,@141=995,@142=5300,
           @143=998,@144=6200,@145=999,@146=7000,@147=1000,@148=8000,
           XYLAST=148,DORC="C",FNARG='RANDMREC',IDNO=1,EXPNORM)

NORMREC   (@101=0,@102=0-3000,@103=12,@104=0-2250,@105=27,@106=0-1930,
           @107=43,@108=0-1720,@109=62,@110=0-1540,@111=84,
           @112=0-1380,@113=104,@114=0-1260,@115=131,@116=0-1120,
           @117=159,@118=0-1000,@119=187,@120=0-890,@121=230,
           @122=0-740,@123=267,@124=0-620,@125=334,@126=0-430,
           @127=432,@128=0-170,@129=500,@130=0,@131=568,@132=170,
           @133=666,@134=430,@135=732,@136=620,@137=770,@138=740,
           @139=813,@140=890,@141=841,@142=1000,@143=869,@144=1120,
           @145=896,@146=1260,@147=916,@148=1380,@149=938,@150=1540,
           @151=957,@152=1720,@153=973,@154=1930,@155=988,@156=2250,
           @157=1000,@158=3000,
           XYLAST=158,DORC="C",FNARG='RANDMREC',IDNO=2,EXPNORM)


DECREC    ('DECIMAL')
SNAREC    ('SNAREF')
TRANSREC  ('TRANSTIM')


REASON1   (CHARS=" THE CONDITIONAL DURATION ENTITY MUST BE STATIONARY"
REASON2   (CHARS=" THE UNITS ARE WRONG")
REASON3   (CHARS=" OF THE COMPLEX INTER-EVENT TIME SPECIFICATION")


ACTNLIST  ('RECLIST',LASTREC=10,@11='NULL')
MOBLIST   ('RECLIST',LASTREC=10,@11='NULL')
STALIST   ('RECLIST',LASTREC=10,@11='NULL')
DSTRLIST  ('RECLIST',LASTREC=10,@11='NULL')
SCSRLIST  ('RECLIST',LASTREC=10,@11='NULL')
MISCLIST  ('RECLIST',LASTREC=10,@11='NULL')
UNITLIST  ('RECLIST',LASTREC=20,@11='ONE',@12='TWO',@13='THREE',
            @14='FOUR',@15='FIVE',@16='SIX',@17='SEVEN',
            @18='EIGHT',@19='NINE',@20='TEN')

SETMEM    (MEPTR(MEM)='MOBLIST',SEPTR(MEM)='STALIST',
           ACPTR(MEM)='ACTNLIST',DISTPTR(MEM)='DSTRLIST',
           SUCPTR(MEM)='SCSRLIST',UNITPTR(MEM)='UNITLIST',
           MISCPTR(MEM)='MISCLIST')

:END OF FILE:
```

A P P E N D I X C

Encoding Rules

for

Producing the English Problem Description

SEMOLOGY FOR ENCODING THE ENGLISH PROBLEM DESCRIPTION:

```
 (1)      ENGLISH(¬QUESTSW(MEM))     -->
                   CLEAR(%SEPTR(MEM),LC=11)
                   CLEAR(%MEPTR(MEM),LC=11)
                   SETAGL(%ACPTR(MEM),LC=11)
                   ACTLIST(%ACPTR(MEM),LC=11)
                   NEWPAR    ENDING
 (2)      ENGLISH    -->    NULL
 (3)      CLEAR(LC.LE.LASTREC)     -->
                   CLEAR(-MARK1(@LC),-MARK2(@LC),LC=LC+1)
 (4)      CLEAR    -->    NULL
 (5)      SETAGL(LC.LE.LASTREC)     -->
                   SET(ASET=@LC(SETAGL)(SETAGL),-MARK1(ASET),
                       SUCCCHANGED(MEM).NE.0,-CHECKED(ASET),-REACHED(ASET))
                   SETAGL(LC=LC+1)
 (6)      SETAGL    -->    NULL(-SUCCCHANGED(MEM))
 (7)      SET(¬MOBENATR(ASET),GOAL(ASET)$'MOBENTY')     -->
                   SET(MOBENATR(ASET)='GOAL')
 (8)      SET(¬MOBENATR(ASET),AGENT(ASET)$'MOBENTY')     -->
                   SET(MOBENATR(ASET)='AGENT')
 (9)      SET    -->    NULL
(10)      ACTLIST(LC.LE.LASTREC)     -->
                   ACTN(%@LC(ACTLIST)(ACTLIST))    ACTLIST(LC=LC+1)
(11)      ACTLIST    -->    NULL
(12)      ACTN(¬MARK1)     -->    NEWPAR    SENT(%ACTN,-PRED,-SUCC)    ACTN(MARK1)
(13)      ACTN(IETM,IETM¬$'ABSTIME')     -->
                   SENT(%ACTN,ATTRIB='IETM',-PRED,-SUCC)    ACTN(-IETM)
(14)      ACTN(LOCATION,¬MARK1(LOCOBJ(LOCATION)))     -->
                   STENDESC(%LOCOBJ(LOCATION(ACTN)),
                    IDSNO=LOCOBJ(LOCATION(ACTN)),MARK1(IDSNO),
                    MBNTY=@MOBENATR(ACTN)(ACTN))        ACTN
(15)      ACTN(DURATION,DURATION¬$'COND',DURATION¬$'ABSTIME')     -->
                   SENT(%ACTN,ATTRIB='DURATION',-PRED,-SUCC)    ACTN(-DURATION)
(16)      ACTN(ASNDISTR)     -->
                   ASNDESC(%ASNDISTR(ACTN))    ACTN(-ASNDISTR)
(17)      ACTN(SUCC)     -->    SUCCDESC(%SUCC(ACTN),PRED=ACTN,-IETM(PRED),
                     -DURATION(PRED),$'ACTION',MARK1(SUCC(ACTN)))
(18)      ACTN    -->    NULL
(19)      ASNDESC     -->
                   RECORD('CMPLX',RP(MEM)=RECORD)
                   RECLOOP1(%ASNDESC,LC=11,ATNO=101,ATNO2=102)
(20)      SUCCDESC($'ACTION')     -->    SENT(%SUCCDESC,-SUCC)
(21)      SUCCDESC('PTYP')     -->
                   RECORD('CMPLX',PRED(SUCCDESC),RP(MEM)=RECORD,
                          CONDITN(SUCCDESC).NE.0,CONDITN(SUCCDESC))
                   RECLOOP2(%SUCCDESC,LC=11,ATNO=102)
(22)      SUCCDESC('FRACTNL')     -->
                   RECORD('CMPLX',PRED(SUCCDESC),RP(MEM)=RECORD,
                          CONDITN(SUCCDESC).NE.0,CONDITN(SUCCDESC))
                   RECLOOP3(%SUCCDESC,LC=11,ATNO=101,ATNO2=102)
(23)      SUCCDESC('QTYP')     -->
                   RECORD('LT',OBJREL=MAXQ(SUCCDESC),RP(MEM)=RECORD)
                   RECORD('AT',LOCOBJ=SUCARG(SUCCDESC),
                    RP2(MEM)=RECORD)
                   RECORD('LINE',LOCATION=RP2(MEM),LENGTH=RP(MEM),
                          ATTRIB='LENGTH',-RP2(MEM),RP(MEM)=RECORD,
                          ¬LOCOBJ(LOCATION),-LOCATION)
                   SUCCDQFS(%SUCCDESC,ACT1=OPENACT,ACT2=CLOSACT)
(24)      SUCCDESC('FTYP'|'STYP')     -->
                   RECORD(%SUCARG(SUCCDESC),RP(MEM)=RECORD,
                    ATTRIB='STATE',STATE='FULL',-LOCATION,
                    ¬STORIND($STRUC),STATE='BUSY')
                   SUCCDQFS(%SUCCDESC,ACT1=CLOSACT,ACT2=OPENACT)
(25)      SUCCDESC     -->    NULL
(26)      SUCCDQFS(¬ACT1,RP(MEM))     -->
                   SUCCDQFS(ACT1=ACT2,-ACT2,NEG(RP(MEM)))
(27)      SUCCDQFS(ACT1$'SUCDSCR1')     -->
                   SUCCDESC(%ACT1(SUCCDQFS),PRED(SUCCDQFS),CONDITN=RP(MEM),
                          -RP(MEM))
                   SUCCDQFS(-ACT1)
(28)      SUCCDQFS(ACT1$'ACTION')     -->
                   SENT(%ACT1(SUCCDQFS),MARK1(ACT1(SUCCDQFS)),-SUCC,
                        PRED(SUCCDQFS),CONDITN=RP(MEM),-RP(MEM),FUTURE,
                        SING)
                   SUCCDQFS(-ACT1)
(29)      SUCCDQFS(ACT2$'SUCDSCR1')     -->
                   SUCCDESC(%ACT2(SUCCDQFS),PRED(SUCCDQFS),CONDITN='OTHERWIS')
(30)      SUCCDQFS(ACT2$'ACTION')     -->
                   SENT(%ACT2(SUCCDQFS),MARK1(ACT2(SUCCDQFS)),-SUCC,
                     -PRED,CONDITN='OTHERWIS',FUTURE,SING)
(31)      SUCCDQFS     -->    NULL
(32)      STENDESC(QUANTITY.GT.1)     -->
                   SENT(%STENDESC,ATTRIB='QUANTITY')
                   STENDESC(-QUANTITY,-LOCATION)
(33)      STENDESC(NUM(CAPACITY).GT.1,NUM(CAPACITY).NE.1000)     -->
                   SENT(%STENDESC,ATTRIB='CAPACITY',-LOCATION)
                   STENDESC(-CAPACITY)
(34)      STENDESC     -->    NULL
(35)      RECLOOP1(ATNO2.LE.XYLAST)     -->
                   RECLOOP1(REC=%@ATNO2,CPC(REC)=NUM(@ATNO),
                    PC(REC)=CPC(REC)-PCPC,PCPC=CPC(REC),
                    ATTRIB(REC)=CLASATR(STRUC(REC)),
```

```
                @LC(SEG)(RP(MEM))=REC,LC=LC+1,ATNO=ATNO+2,
                ATNO2=ATNO2+2,ATTRIB(REC).EQ.'SOUP',SOUP(REC)=SUP(REC))
(36)    RECLOOP1   -->   RECLOOP(%RECLOOP1)
(37)    RECLOOP2(ATNO.LE.XYLAST)   -->
               RECLOOP2(REC=%@ATNO,MARK1(@ATNO),-SUCC(REC),
                @LC(SEG)(RP(MEM))=REC,LC=LC+1,ATNO=ATNO+2)
(38)    RECLOOP2   -->   RECLOOP(%RECLOOP2)
(39)    RECLOOP3(ATNO2.LE.XYLAST)   -->
               RECLOOP3(REC=%@ATNO2,MARK1(@ATNO2),-SUCC(REC),
                REC2=%@MOBENATR(REC)(REC),CPC(REC2)=NUM(@ATNO),
                PC(REC2)=CPC(REC2)-PCPC,PCPC=CPC(REC2),
                @MOBENATR(REC)(REC)=REC2,@LC(SEG)(RP(MEM))=REC,
                LC=LC+1,ATNO=ATNO+2,ATNO2=ATNO2+2)
(40)    RECLOOP3   -->   RECLOOP(%RECLOOP3)
(41)    RECLOOP   -->
               SENT(%RP(MEM),-RP(MEM),LR=LC(RECLOOP)-1,LC=11)
(42)    RECORD   -->   NULL
(43)    ENDING   -->   SENT(%MEM,ATTRIB='PROBTIME',-SUCC,-PRED)

LEXOLOGY FOR ENCODING ENGLISH:

(44)    SENT(¬QUESMK,INTRGPH|INTERG|¬@ATTRIB)   -->   SENT(QUESMK)
(45)    SENT(SUCC)   -->
               CONJ('BEFORE')   PRPTCL(%SUCC(SENT))   ,   SENT(-SUCC)
(46)    SENT(PRED)   -->
               CONJ('AFTER')   PRPTCL(%PRED(SENT))   ,   SENT(-PRED)
(47)    SENT(CONDITN.EQ.'OTHERWIS')   -->
               CONJ('OTHERWIS')   ,   SENT(-CONDITN)
(48)    SENT(CONDITN)   -->
               CONJ('IF')   FINCL(%CONDITN(SENT))   ,   SENT(-CONDITN)
(49)    SENT('CMPLX',LC.LT.LR)   -->
               FINCL(%@LC(SENT)(SENT))   ,   SENT(LC=LC+1)
(50)    SENT('CMPLX')   -->   CONJ('AND')   FINCL(%@LC(SENT)(SENT))
               SENT1(%SENT)
(51)    SENT   -->   FINCL(%SENT)   SENT1(%SENT)
(52)    SENT1(REASON)   -->   ,   CONJ('BECAUSE')
               PHRASE(CHARS=REASON(SENT1))   SENT1(-REASON)
(53)    SENT1(QUESMK)   -->   ?   #
(54)    SENT1   -->   .   #
(55)    FINCL(ATTRIB)   -->   ATVCL(%FINCL)
(56)    FINCL   -->   VERBPH(%FINCL)
(57)    PRPTCL   -->   VERBPH(%PRPTCL,PRESPART)
(58)    INFINCL   -->   VERBPH(% INFINCL,INFIN)
(59)    ATVCL(¬@ATTRIB)   -->   ATVQUES(%ATVCL)
(60)    ATVCL(ATTRIB.EQ.'DURATION',DURATION$'COND')   -->   VERBPH(%ATVCL)
(61)    ATVCL(ATTRIB.EQ.'PROBTIME')   -->
               PHRASE(CHARS=" THE SIMULATION IS TO BE RUN FOR")
               VALUE(VAL=PROBTIME(MEM))   ,
               PHRASE(CHARS=" USING A BASIC TIME UNIT OF")
               VALUE(VAL=TIMUNIT(MEM))
(62)    ATVCL(ATTRIB.EQ.'QUANTITY')   -->
               ADV('THERE')
               VERB('BE',PLUR,QUANTITY(ATVCL).EQ.1,-PLUR,SING)
               VALUE(VAL=QUANTITY(ATVCL))
               NOUNPH(%ATVCL,-ATTRIB,PLUR,PRM,QUANTITY.EQ.1,-PLUR,SING)
(63)    ATVCL(ATTRIB.EQ.'CAPACITY')   -->
               FINCL('BE',PREDVAL=CAPACITY(ATVCL),SUBJECT=ATVCL,
                      MODAL=MODAL(ATVCL),NEG=NEG(ATVCL))
               NOUN(SUP(MBNTY(ATVCL)),PLUR)
(64)    ATVCL(ATTRIB.EQ.'TYPE')   -->
               RECORD(SUP(ATVCL),TYPE(ATVCL),PLUR,PRM,
                RP(MEM)=RECORD)
               FINCL('BE',PREDNOM=RP(MEM),-RP(MEM),SUBJECT=ATVCL,
                      MODAL=MODAL(ATVCL),NEG=NEG(ATVCL),
                      -@ATTRIB(ATVCL)(SUBJECT),-ATTRIB(SUBJECT))
(65)    ATVCL(ATTRIB.EQ.'SOUP')   -->
               RECORD(%ATVCL,-PC,-CPC,-ATTRIB,PLUR,PRM,RP(MEM)=RECORD)
               FINCL('BE',PREDNOM=RP(MEM),-RP(MEM),SUBJECT=ATVCL,
                      MODAL=MODAL(ATVCL),NEG=NEG(ATVCL),
                      SUP(SUBJECT)=SUP(STRUC(SUBJECT)),-ATTRIB(SUBJECT))
(66)    ATVCL(@ATTRIB$'QUALVAL')   -->
               FINCL('BE',PREDADJ=@ATTRIB(ATVCL)(ATVCL),SUBJECT=ATVCL,
                      MODAL=MODAL(ATVCL),NEG=NEG(ATVCL),
                      -@ATTRIB(ATVCL)(SUBJECT),-ATTRIB(SUBJECT))
(67)    ATVCL   -->
               FINCL('BE',PREDVAL=@ATTRIB(ATVCL)(ATVCL),SUBJECT=ATVCL,
                      MODAL=MODAL(ATVCL),NEG=NEG(ATVCL))
(68)    ATVQUES(INTRGPH(ATTRIB))   -->   FINCL(%ATVQUES,INTRGPH(ATTRIB),
                                                -ATTRIB)
(69)    ATVQUES   -->   FINCL('BE',INTRGPH=" WHAT",SUBJECT=ATVQUES)
(70)    RECORD   -->   NULL
(71)    VALUE(SUP(VAL).EQ.'UNIT')   -->   ADJ(NUM(VAL(VALUE)))
(72)    VALUE(VAL$'QUANVAL')   -->   NOUNPH(%VAL(VALUE))
(73)    VALUE(VAL$'RELIND1')   -->   COMPPH(SUP(VAL(VALUE)))
                                      OBJCOMP(%OBJREL(VAL(VALUE)))
(74)    VALUE(VAL$'STDIST')   -->
               ADV(SUP(VAL(VALUE)))   VERB('DISTRIBU',PASTPART)
               ATVPH1(%VAL(VALUE),ATTRIB='MEAN')
               ATVPH2(%VAL(VALUE),ATTRIB='RANGE')
               ATVPH2(%VAL(VALUE),ATTRIB='STDEV')
(75)    VALUE(SUP(VAL).EQ.'TYPTABL')   -->
               VALOOP(%VAL(VALUE),ATNO=101,ATNO2=102)
(76)    VALUE   -->   ADJ(NUM=VAL(VALUE))
(77)    COMPPH('LT')   -->   PHRASE(CHARS=" LESS THAN")
```

```
(78)    COMPPH('EQ')    -->   PHRASE(CHARS=" EQUAL TO")
(79)    COMPPH('GT')    -->   PHRASE(CHARS=" GREATER THAN")
(80)    OBJCOMP($'QUANVAL')    -->   VALUE(VAL=OBJCOMP)
(81)    OBJCOMP($'ENTITY')    -->   NOUNPH(%OBJCOMP)
(82)    ATVPH1(@ATTRIB)    -->    ,   PREP('WITH')    ATVPH3(%ATVPH1)
(83)    ATVPH1             -->    NULL
(84)    ATVPH2(@ATTRIB,¬MEAN)    -->    ,   PREP('WITH')    ATVPH3(%ATRPH2)
(85)    ATVPH2(@ATTRIB)    -->    CONJ('AND')    ATVPH3(%ATVPH2)
(86)    ATVPH2             -->    NULL
(87)    ATVPH3         -->    ADJ('A')    NOUN(SUP=ATTRIB(ATVPH3))
                  ATVPH4(VAL=@ATTRIB(ATVPH3)(ATVPH3))
                  VALUE(VAL=@ATTRIB(ATVPH3)(ATVPH3))
(88)    ATVPH4(VAL$'STDIST')    -->    PHRASE(CHARS=" WHICH IS")
(89)    ATVPH4    -->    PREP('OF')
(90)    VALOOP(ATNO2.LT.XYLAST)    -->    VALOOP2(%VALOOP)    ,
                  VALOOP(ATNO=ATNO+2,ATNO2=ATNO2+2)
(91)    VALOOP        -->    CONJ('AND')    VALOOP2(%VALOOP)
(92)    VALOOP2     -->    VALUE(VAL=@ATNO2(VALOOP2)(VALOOP2))
                  PREPPH('FOR',OBJPPH=@ATNO(VALOOP2)(VALOOP2),
                   PLUR(OBJPPH))
(93)    VERBPH($'ENTITY')    -->    VERBPH('BE',SUBJECT=VERBPH)
(94)    VERBPH(¬SUBJECT,MOBENATR.EQ.'AGENT')    -->
                  VERBPH(SUBJECT=AGENT,-AGENT)
(95)    VERBPH(¬SUBJECT,MOBENATR.EQ.'GOAL')    -->
                  VERBPH(SUBJECT=GOAL,-GOAL,PASSIVE)
(96)    VERBPH(¬SUBJECT)    -->    VERBPH(SUBJECT='SOMETHING')
(97)    VERBPH(¬NUMB,SUBJECT$'MOBENTY')    -->    VERBPH(PLUR)
(98)    VERBPH(¬NUMB,QUANTITY(SUBJECT).GT.1)    -->    VERBPH(PLUR)
(99)    VERBPH(¬NUMB)    -->    VERBPH(SING)
(100)   VERBPH(INTRGPH)    -->    PHRASE(CHARS=INTRGPH(VERBPH))
                  VERBPH(INTERG,-INTRGPH)
(101)   VERBPH(PRESPART)    -->    VERBPH2(%VERBPH,-SUBJECT,-NUMB)
(102)   VERBPH(INFIN)    -->
                  PREP('FOR')    NOUNPH(%SUBJECT(VERBPH),NUMB=
                   NUMB(VERBPH),$'ENTITY',¬IDSNO,IDSNO=SUBJECT(VERBPH))
                  PREP('TO')    VERBPH2(%VERBPH,-SUBJECT,-NUMB)
(103)   VERBPH    -->    VERBPH2(%VERBPH,NUMBSUBJ=NUMB)
(104)   VERBPH2    -->
                  VERBPH3(%VERBPH2)    VERBPH4(%VERBPH2)
(105)   VERBPH3('GOTO')    -->    VERBPH3('GO')
(106)   VERBPH3(¬'BE',¬AUXIL,FINITE,INTERG|NEG)    -->    VERBPH3(DO)
(107)   VERBPH3(NEG,¬SUBJECT,¬FINITE)    -->    ADV('NOT')    VERBPH3(-NEG)
(108)   VERBPH3(MODAL,INTERG|¬SUBJECT)    -->
                  VERB(MODAL=MODAL(VERBPH3),VFORM=VFORM(VERBPH3))
                  VERBPH3(-MODAL,-VFORM,-INTERG,INFIN)
(109)   VERBPH3(PERFECT,INTERG|¬SUBJECT)    -->
                  VERB('HAV',VFORM=VFORM(VERBPH3))
                  VERBPH3(-PERFECT,-VFORM,-INTERG,PASTPART)
(110)   VERBPH3(PROG,INTERG|¬SUBJECT)    -->
                  VERB('BE',VFORM=VFORM(VERBPH3))
                  VERBPH3(-PROG,-VFORM,-INTERG,PRESPART)
(111)   VERBPH3(PASSIVE,INTERG|¬SUBJECT)    -->
                  VERB('BE',VFORM=VFORM(VERBPH3))
                  VERBPH3(-PASSIVE,-VFORM,-INTERG,PASTPART)
(112)   VERBPH3(SUBJECT,¬INTERG)    -->
                  NOUNPH(%SUBJECT(VERBPH3),NUMB=NUMBSUBJ(VERBPH3),
                        $'ENTITY',¬IDSNO,IDSNO=SUBJECT(VERBPH3))
                  VERBPH3(-SUBJECT)
(113)   VERBPH3('BE',FINITE,INTERG,SUBJECT)    -->
                  VERBPH3(-INTERG,-SUBJECT)
                  NOUNPH(%SUBJECT(VERBPH3),NUMB=NUMBSUBJ(VERBPH3),
                        $'ENTITY',¬IDSNO,IDSNO=SUBJECT(VERBPH3))
(114)   VERBPH3('BE',FINITE,NEG)    -->    VERBPH3(-NEG)    ADV('NOT')
(115)   VERBPH3    -->    VERB(SUP(VERBPH3),VFORM=VFORM(VERBPH3))
(116)   VERBPH4('GOTO')    -->
                  PHRASE(CHARS=" TO AN APPROPRIATE")
                  NOUNPH(%LOCOBJ(LOCATION(VERBPH4)),-LOCATION,PRM)
                  PHRASE(CHARS=" WITH THE SHORTEST LINE")
(117)   VERBPH4(AGENT)    -->
                  PREPPH('BY',OBJPPH=AGENT(VERBPH4))
                  VERBPH4(-AGENT)
(118)   VERBPH4(GOAL)    -->
                  NOUNPH(%GOAL(VERBPH4))
                  VERBPH4(-GOAL)
(119)   VERBPH4(LOCATION,SUP(LOCOBJ(LOCATION)).EQ.'PARAMNO')    -->
                  PRON('THERE')    VERBPH4(-LOCATION)
(120)   VERBPH4(LOCATION,'ENTER'|'LEAV')    -->
                  NOUNPH(%LOCOBJ(LOCATION(VERBPH4)))    VERBPH4(-LOCATION)
(121)   VERBPH4(LOCATION)    -->
                  PREPPH(%LOCATION(VERBPH4),OBJPPH=LOCOBJ)
                  VERBPH4(-LOCATION)
(122)   VERBPH4(IETM$'ABSTIME')    -->
                  PREPPH('EVERY',OBJPPH=IETM(VERBPH4))
(123)   VERBPH4(DURATION$'ABSTIME')    -->
                  PREPPH('FOR',OBJPPH=DURATION(VERBPH4))
(124)   VERBPH4(DURATION$'COND')    -->
                  CONJ('UNTIL')    FINCL(%CONDENTY(DURATION(VERBPH4)),
                   ATTRIB='STATE',STATE='AVAILABL',-LOCATION,
                   SUP(DURATION(VERBPH4)).NE.'COND1',
                   STATE='UNAVAIL')
(125)   VERBPH4(PREDADJ)    -->    ADJ(SUP=PREDADJ(VERBPH4))
(126)   VERBPH4(PREDNOM)    -->    NOUNPH(%PREDNOM(VERBPH4))
(127)   VERBPH4(PREDVAL)    -->    VALUE(VAL=PREDVAL(VERBPH4))
(128)   VERBPH4          -->    NULL
(129)   PREPPH     -->
```

```
                  PREP(SUP(PREPPH))     NOUNPH(%OBJPPH(PREPPH))
(130)    NOUNPH(PC,PC.EQ.CPC)    -->
                  ADJ(NUM=PC(NOUNPH)/10)     NOUN('PERCENT')
                  PREPPH('OF',OBJPPH=NOUNPH,-PC(OBJPPH),
                   -COLOR(OBJPPH),-TYPE(OBJPPH),PLUR(OBJPPH))
(131)    NOUNPH(PC,CPC.GT.990,¬NOREST(MEM))    -->    PHRASE(CHARS=" THE REST")
(132)    NOUNPH(PC)    -->    ADJ(NUM=PC(NOUNPH)/10)     NOUN('PERCENT')
(133)    NOUNPH($'QUANVAL')    -->
                  ADJ(NUM(NOUNPH))    NOUN(%NOUNPH,PLUR,NUM.EQ.1,-PLUR,SING)
(134)    NOUNPH(¬PRM,'CARGO')    -->    NOUNPH(PRM)
(135)    NOUNPH(¬PRM,$'STATENTY',QUANTITY.GT.1)    -->    ADJ('A')    NOUNPH(PRM)
(136)    NOUNPH(¬PRM)    -->    ADJ('THE')     NOUNPH(PRM)
(137)    NOUNPH(ATTRIB.EQ.'IETM')    -->    NOUN('TIME')
                  PREPPH('BETWEEN',OBJPPH=NOUNPH,-ATTRIB(OBJPPH),PLUR(OBJPPH))
(138)    NOUNPH(ATTRIB.EQ.'DURATION')    -->    NOUN('TIME')
                  INFINCL(%NOUNPH,-CAN,-NEG,-INTERG,-NUMB)
(139)    NOUNPH(ATTRIB)    -->    NOUN(SUP=ATTRIB(NOUNPH))
                  PREPPH('OF',OBJPPH=NOUNPH,-@ATTRIB(NOUNPH)(OBJPPH),
                   -ATTRIB(OBJPPH),-PRM(OBJPPH))
(140)    NOUNPH(SIZE$'RELSIZ')    -->
                  ADJ(SUP=SIZE(NOUNPH))    NOUNPH(-SIZE)
(141)    NOUNPH(COLOR)    -->
                  ADJ(SUP=COLOR(NOUNPH))    NOUNPH(-COLOR)
(142)    NOUNPH(TYPE)    -->
                  ADJ(SUP=TYPE(NOUNPH))   NOUNPH(-TYPE)
(143)    NOUNPH(LOCATION)    -->    NOUNPH(-LOCATION)
                  PREPPH(%LOCATION(NOUNPH),OBJPPH=LOCOBJ)
(144)    NOUNPH(MOBENATR)    -->    NOUNPH(-MOBENATR)
                  PREPPH('OF',OBJPPH=%@MOBENATR(NOUNPH)(NOUNPH),PLUR(OBJPPH))
(145)    NOUNPH($'STDIST')    -->    ADJ(SUP(NOUNPH))    NOUN('DISTRIBU')
(146)    NOUNPH    -->    NOUN(%NOUNPH)
(147)    VERB(FUTURE)    -->    VERB('WILL',-FUTURE,-VFORM)
(148)    VERB(CAN)    -->    VERB('CAN',-CAN,-VFORM)
(149)    VERB(DO)    -->    VERB('DO',-DO)
```

MORPHOLOGY FOR ENCODING ENGLISH:

```
(150)    VERB    -->    #    VERBP(%VERB,E*,EN*)
(151)    NOUN    -->    #    NOUNP(%NOUN)
(152)    ADV    -->    #    ADVP(%ADV)
(153)    ADJ    -->    #    ADJP(%ADJ)
(154)    PREP    -->    #    PREPP(%PREP)
(155)    CONJ    -->    #    CONJP(%CONJ)
(156)    PRON    -->    #    PRONP(%PRON)
(157)    VERBP('BE',SING)    -->    I   S
(158)    VERBP('BE',PLUR)    -->    A   R   E
(159)    VERBP('HAV',SING)    -->    H   A   S
(160)    VERBP(PRESPART)    -->    VERBS(SUP(VERBP))    I   N   G
(161)    VERBP(PASTPART,EN)    -->    VERBS(SUP(VERBP))    E   N
(162)    VERBP(PASTPART)    -->    VERBS(SUP(VERBP))    E   D
(163)    VERBP(E,SING)    -->    VERBS(SUP(VERBP))   E   S
(164)    VERBP(E,PLUR|INFIN)    -->    VERBS(SUP(VERBP))    E
(165)    VERBP(SING)    -->    VERBS(SUP(VERBP))    S
(166)    NOUNP('MAN',PLUR)    -->    M   E   N
(167)    NOUNP(PLUR)    -->    NOUNS(SUP(NOUNP))    S
(168)    NOUNP    -->    NOUNS(SUP(NOUNP))
(169)    VERBS('DISTRIBU')    -->    STEM(CHARS="DISTRIBUT")
(170)    NOUNS('ARRIV')    -->    VERBS('ARRIV')    A   L
(171)    NOUNS('ENTER')    -->    STEM(CHARS="ENTRIE")
(172)    NOUNS($'ACTION')    -->    VERBP(SUP(NOUNS),PRESPART)
(173)    NOUNS('DISTRIBU')    -->    WORD(CHARS="DISTRIBUTION")
(174)    NOUNS(NAME.EQ."SOMETHIN")    -->    WORD(CHARS="SOMETHING")
(175)    NOUNS('GASSTA')    -->    NAME(CHARS="GAS STATION")
(176)    NOUNS('RANGE')    -->    NAME(CHARS="HALF-RANGE")
(177)    NOUNS('STDEV')    -->    NAME(CHARS="STANDARD DEVIATION")
(178)    ADVP(LY*)    -->    ADJP(SUP(ADVP))    L   Y
(179)    ADJP(NUM)    -->    NUMBER(%ADJP)
(180)    ADJP('EXPON')    -->    WORD(CHARS="EXPONENTIAL")
(181)    ADJP('COMMERCE')    -->    WORD(CHARS="COMMERCIAL")
(182)    ADJP('MISC')    -->    WORD(CHARS="MISCELLANEOUS")
(183)    ADJP('UNAVAIL')    -->    U   N   ADJP('AVAILABL')
(184)    ADJP('AVAILABL')    -->    WORD(CHARS="AVAILABLE")
(185)    CONJP('OTHERWIS')    -->    WORD(CHARS="OTHERWISE")
(186)    NEWPAR    -->    OUTPUT(@11=1,@12=7)
(187)    PHRASE    -->    OUTPUT(@13=CHARS(PHRASE))
(188)    WORD    -->    OUTPUT(@13=CHARS(WORD))
(189)    STEM    -->    OUTPUT(@13=CHARS(STEM))
(190)    NAME    -->    OUTPUT(@13=CHARS(NAME))
(191)    NUMBER(¬NUM)    -->    0
(192)    NUMBER    -->    OUTPUT(@14=NUM(NUMBER))
(193)    DECNUMB(¬NUM)    -->    0.0
(194)    DECNUMB    -->    OUTPUT(@15=NUM(DECNUMB))
(195)    NULL    -->    OUTPUT
```

:END OF FILE:

A P P E N D I X D

Encoding Rules

for

Producing the GPSS Program

SEMOLOGY FOR ENCODING GPSS PROGRAM:

```
(201)      GPSSPROG(¬QUESTSW(MEM))    -->    INITIALGPSS
                     SETAGL(%ACPTR(MEM),LC=11)
                     STMNT('SIMULATE')    STMNT(%'SEEDS')
                     SELIST(%SEPTR(MEM),LC=11)
                     MELIST(%MEPTR(MEM),LC=11)     EXPFUNC     NORMFUNC
                     DISTLIST(%DISTPTR(MEM),LC=11)
                     SUCLIST(%SUCPTR(MEM),LC=11)
                     DSTLIST(%DISTPTR(MEM),LC=11)
                     ACLIST(%ACPTR(MEM),LC(MEM)=11)     TMLOOP
(202)      GPSSPROG   -->   NULL
(203)      INITIALGPSS    -->    NULL(GRNNO(MEM)=0)

(204)      SELIST(LC.LE.LASTREC)    -->
                     STENTITY(%@LC(SELIST)(SELIST),IPDP=@LC(SELIST)(SELIST))
                     SELIST(LC=LC+1)
(205)      SELIST   -->   NULL
(206)      MELIST(LC.LE.LASTREC)    -->    MENTITY(%@LC(MELIST)(MELIST))
                     MELIST(LC=LC+1)
(207)      MELIST   -->   NULL
(208)      STENTITY    -->
                     EQUCARD(%STENTITY)
                     STMNT('STORAGE',LABL=IDNO(STENTITY),
                            ARGA=QUANTITY($STRUC(STENTITY))
                            *NUM(CAPACITY($STRUC(STENTITY))),
                            IPDP(STENTITY))
(209)      MENTITY    -->    EQUCARD(%MENTITY)    TABLECARD(%MENTITY)
(210)      EQUCARD(IDNAME,¬CLASATR)    -->
                     STMNT('EQU',LABL=IDNAME(EQUCARD),ARGA=IDNO(EQUCARD),
                            ARGB="S",ARGC="Q",¬STORIND($STRUC(EQUCARD)),ARGB="F",
                            EQUCARD¬$'STATENTY',ARGB="T",-ARGC)
(211)      EQUCARD --> NULL
(212)      TABLECARD(¬CLASATR)    -->
                     STMNT('TABLE', LABL=IDNO(TABLECARD),
                            ARGA="M1",ARGB=LOWINT($STRUC(TABLECARD)),
                            ARGC=INTWIDTH($STRUC(TABLECARD)),
                            ARGD=NUMINT($STRUC(TABLECARD)))
(213)      TABLECARD --> NULL

(214)      EXPFUNC(EXPUSED(MEM))   -->    FNDEF(%'EXPONREC',IPDP='EXPONREC')
(215)      EXPFUNC    -->    NULL
(216)      NORMFUNC(NORMUSED(MEM))    -->    FNDEF(%'NORMREC',IPDP='NORMREC')
(217)      NORMFUNC    -->    NULL
(218)      DISTLIST(LC.LE.LASTREC)    -->
                     FNDEF(%@LC(DISTLIST)(DISTLIST),
                            IPDP=@LC(DISTLIST)(DISTLIST))
                     DISTLIST(LC=LC+1)
(219)      DISTLIST    -->    NULL
(220)      SUCLIST(LC.LE.LASTREC)    -->
                     FNDEF(%@LC(SUCLIST)(SUCLIST),IPDP=@LC(SUCLIST)(SUCLIST))
                     SUCLIST(LC=LC+1)
(221)      SUCLIST    -->    NULL
(222)      FNDEF(XYLAST)     -->
                     STMNT('FUNCTION', LABL=IDNO(FNDEF),
                            ARGA=FNARG(FNDEF),TEMP=XYLAST(FNDEF)-100,
                            ARGB=%'ARGREC',FNTYPE(ARGB)=DORC(FNDEF),
                            NOPTS(ARGB)=TEMP/2,IPDP(FNDEF))
                     STMNT(%FNDEF,LC=101)
(223)      FNDEF    -->    NULL

(224)      DSTLIST(LC.LE.LASTREC)    -->
                     DISTDEF(%@LC(DSTLIST)(DSTLIST))    DSTLIST(LC=LC+1)
(225)      DSTLIST    -->    NULL
(226)      DISTDEF('NORMAL')    -->
                     STMNT('FVARIABL',LABL=IDNO(DISTDEF),
                            ARGA=%'ARGREC',MEAN(ARGA)=MEAN(DISTDEF),
                            STDEV(ARGA)=STDEV(DISTDEF))
(227)      DISTDEF    -->    NULL

(228)      ACLIST(LC(MEM).LE.LASTREC)    -->
                     ACTT(%@LC(MEM)(ACLIST),IPDP(MEM)=@LC(MEM)(ACLIST))
                     ACLIST(LC(MEM)=LC(MEM)+1)
(229)      ACLIST    -->    NULL
(230)      ACTT -->  STMNT('COMMENT',SENTT=ACTT)     ACT(%ACTT)
(231)      ACT(¬PRED,'ARRIV'|'ENTER')    -->
                     STMNT('GENERATE',XMOD=@MOBENATR(ACT)(ACT),ARGA=IETM(ACT))
                     STMNT('ASSIGN',XMOD=2,ARGA=1,ARGB=ASNDISTR(ACT),¬ARGB,
                            ARGB=@MOBENATR(ACT)(ACT))
                     STMNT('ENTER',ARGA=LOCOBJ(LOCATION(ACT)))
                     SUCSTMNT(%SUCC(ACT),IPDP=SUCC(ACT))
(232)      ACT($'ACTIVITY', STORIND($STRUC(LOCOBJ(LOCATION))))   -->
                     STMNT('QUEUE',LABL=ACT,ARGA=LOCOBJ(LOCATION(ACT)))
                     STMNT('ENTER',ARGA=LOCOBJ(LOCATION(ACT)),
                            ARGB=CONSUMP($STRUC(@MOBENATR(ACT)(ACT))),
                            NUM(ARGB).EQ.1,-ARGB)
                     STMNT('DEPART',ARGA=LOCOBJ(LOCATION(ACT)))
                     STMNT('ADVANCE',ARGA=DURATION(ACT))
                     STMNT('LEAVE', ARGA=LOCOBJ(LOCATION(ACT)),
```

```
                    ARGB=CONSUMP($STRUC(@MOBENATR(ACT)(ACT))),
                    NUM(ARGB).EQ.1,-ARGB)
               SUCSTMNT(%SUCC(ACT),IPDP=SUCC(ACT))
(233)     ACT($'ACTIVITY')    -->
               STMNT('QUEUE',LABL=ACT,ARGA=LOCOBJ(LOCATION(ACT)))
               STMNT('SEIZE',ARGA=LOCOBJ(LOCATION(ACT)))
               STMNT('DEPART',ARGA=LOCOBJ(LOCATION(ACT)))
               STMNT('ADVANCE',ARGA=DURATION(ACT))
               STMNT('RELEASE',ARGA=LOCOBJ(LOCATION(ACT)))
               SUCSTMNT(%SUCC(ACT),IPDP=SUCC(ACT))
(234)     ACT('GOTO')    -->
               STMNT('SELECT',LABL=ACT,MOD="TMIN",ARGA=NUM(ARGAZ(ACT)),
               ARGB(ARGAZ(ACT)),ARGC(ARGAZ(ACT)),
               ARGE(ARGAZ(ACT)))
               SUCSTMNT(%SUCC(ACT),IPDP=SUCC(ACT))
(235)     ACT('LEAV',¬SUCC)    -->
               STMNT('TABULATE',LABL=ACT,ARGA='TRANSREC',
                    ARGB='PARAM1',ARGC=0-1)
               STMNT('LEAVE', ARGA=LOCOBJ(LOCATION(ACT)))
               STMNT('TERMINAT')
(236)     ACT    -->    NULL
(237)     SUCSTMNT($'QTYP')    -->
               STMNT('TEST',MOD="L",ARGA=%'SNAREC',NAM(ARGA)="Q",
                    NAM2(ARGA)=SUCARG(SUCSTMNT),ARGB=MAXQ(SUCSTMNT),
                    ARGC=CLOSACT(SUCSTMNT))
               STMNT('TRANSFER',ARGB=OPENACT(SUCSTMNT))
(238)     SUCSTMNT($'FTYP'|$'STYP')    -->
               STMNT('GATE',MOD="SNF",ARGA=SUCARG(SUCSTMNT),
                    ARGB=CLOSACT(SUCSTMNT),
                    ¬STORIND($STRUC(SUCARG(SUCSTMNT))),MOD="NU")
               STMNT('TRANSFER',ARGB=OPENACT(SUCSTMNT))
(239)     SUCSTMNT    -->    STMNT('TRANSFER',ARGB=IPDP(SUCSTMNT))


(240)     TMLOOP    -->
               STMNT('COMMENT',CHARS=" TIMING LOOP",-IPDP(MEM))
               STMNT('GENERATE',ARGA=PROBTIME(MEM))
               STMNT('TERMINAT',ARGA=1)
               STMNT('START',ARGA=1)
               STMNT('END')


(241)     STMNT(¬STORIND($STRUC(ARGA)), 'ENTER'|'LEAVE')    -->  NULL
(242)     STMNT('TRANSFER',ARGB.EQ.@LC(MEM)(ACPTR(MEM)))    -->    NULL
(243)     STMNT(ARGA$'UNIFORM')    -->
               STMNT(ARGB=RANGE(ARGA),ARGA=MEAN(ARGA))
(244)     STMNT(ARGA$'EXPON')    -->
               STMNT(ARGA=MEAN(ARGA),ARGB=%'SNAREC',NAM(ARGB)="FN",
                    NUM(ARGB)=1)
(245)     STMNT(ARGA$'COND')    -->
               STMNT('GATE',TARGA=ARGA,
                    MOD=SMOD($(TARGA)),ARGA=CONDENTY(TARGA),
                    ¬STORIND($STRUC(ARGA)),MOD=FMOD($(TARGA)),-TARGA)
(246)     STMNT    -->    GSTMNT(%STMNT)


LEXOLOGY FOR ENCODING GPSS:

(247)     GSTMNT(XYLAST)    -->    XYOUT(%GSTMNT)
(248)     GSTMNT('STORAGE',ARGA.EQ.1)    -->    NULL
(249)     GSTMNT('TABULATE',ARGC)    -->
               GSTMNT(ARGA=ARGB,ARGB=ARGC,-ARGC,ARGB.LT.0,-ARGB)
(250)     GSTMNT('COMMENT')    -->    NEWLINE1  *  NEWLINE1  *  COLUMN7
               COMMENT(%GSTMNT)
(251)     COMMENT(CHARS)    -->    PHRASE(CHARS(COMMENT))
(252)     COMMENT(SENTT)    -->
               SENT(%SENTT(COMMENT),-PRED,-SUCC,-CONDITN,-IETM,
                  -DURATION,LOCATION=%LOCATION,
                  LOCOBJ(LOCATION)=%LOCOBJ(LOCATION),
                  -LOCATION(LOCOBJ(LOCATION)))
(253)     GSTMNT    -->    NEWLINE2     LABFLD(LABL(GSTMNT))
                           COLUMN8      OPFLD(SUP(GSTMNT))
                           COLUMN13     MODFLD(MOD(GSTMNT))
                           COLUMN19
                                   ARGFLD(APTR=ARGA(GSTMNT),COMCNT(MEM)=0)
                                   ARGFLD(APTR=ARGB(GSTMNT),NCOM=1)
                                   ARGFLD(APTR=ARGC(GSTMNT),NCOM=2)
                                   ARGDH(%GSTMNT)
(254)     ARGDH('RMULT'|'SELECT'|'TABLE')    -->
                                   ARGFLD(APTR=ARGD(ARGDH),NCOM=3)
                                   ARGFLD(APTR=ARGE(ARGDH),NCOM=4)
                                   ARGFLD(APTR=ARGF(ARGDH),NCOM=5)
                                   ARGFLD(APTR=ARGG(ARGDH),NCOM=6)
                                   ARGFLD(APTR=ARGH(ARGDH),NCOM=7)
(255)     ARGDH    -->    NULL

(256)     LABFLD(LABL)    -->    ARG(DATA=LABL(LABFLD))
(257)     LABFLD    -->    NULL
(258)     MODFLD(MOD)    -->    ARG(DATA=MOD(MODFLD))
(259)     MODFLD    -->    NULL
(260)     ARGFLD(APTR)    -->    COMMAS(NCOM(ARGFLD))  ARG(DATA=APTR(ARGFLD))
(261)     ARGFLD    -->    NULL
(262)     XYOUT    -->    NEWLINE1
               LINE(%XYOUT,LAST=LC+7,TLAST=XYLAST-LC,
                    TLAST.LT.7,LAST=LC+TLAST)
```

```
(263)      LINE(LC.LE.LAST)    -->
                      ARG(DATA=%'DECREC',NUM(DATA)=@LC(LINE)(LINE),
                           LC(LINE)=LC(LINE)+1,¬EXPNORM(LINE),
                           DATA=NUM(DATA))        ,
                      ARG(DATA=%'DECREC',NUM(DATA)=@LC(LINE)(LINE),¬EXPNORM(LINE),
                           DATA=NUM(DATA))   /    LINE(LC=LC+1)
(264)      LINE(LC.LE.XYLAST)     -->     XYOUT(%LINE)
(265)      LINE     -->    NULL
```

MORPHOLOGY FOR ENCODING GPSS:

```
(266)      OPFLD('TERMINAT')    -->    NAME(CHARS="TERMINATE")
(267)      OPFLD('FVARIABL')    -->    NAME(CHARS="FVARIABLE")
(268)      ARG(@9=DATA,GETYPE,@9.EQ.3)    -->    PARG(%DATA(ARG))
(269)      ARG(@9.EQ.0)    -->    NUMBER(NUM=DATA(ARG))
(270)      ARG    -->    NAME(CHARS=DATA(ARG))

(271)      PARG($'ACTION')     -->    A    C    T    NUMBER(NUM=IDNO(PARG))
(272)      PARG($'ENTITY')   -->   NAME(CHARS=IDNAME(PARG))
(273)      PARG('DECIMAL')    -->    DECNUMB(NUM(PARG))
(274)      PARG($'ABSTIME')    -->    TIMARG(%PARG)
(275)      PARG($'QUANVAL')    -->    NUMBER(NUM(PARG))
(276)      PARG('SNAREF',NUM)     -->
                  NAME(CHARS=NAM(PARG))
                  NUMBER(NUM(PARG))
(277)      PARG('SNAREF')    -->    NAME(CHARS=NAM(PARG))     $
                  NAME(CHARS=IDNAME(NAM2(PARG)))
(278)      PARG('PARAMNO')    -->    P    NUMBER(NUM(PARG))
(279)      PARG('TRANSTIM')    -->    M    1
(280)      PARG('RANDM')    -->
                  R    N    NUMBER(GRNNO(MEM)=GRNNO(MEM)+1,NUM=GRNNO(MEM),
                                 GRNNO(MEM).EQ.8,-GRNNO(MEM))
(281)      PARG('NORMAL')    -->    V    NUMBER(NUM=IDNO(PARG))
(282)      PARG(MEAN,STDEV)    -->    ARG(DATA=MEAN(PARG))    +
                  ARG(DATA=STDEV(PARG))    *    F    N    2
(283)      PARG(XYLAST)    -->    F    N    NUMBER(NUM=IDNO(PARG))
(284)      PARG(NOPTS)    -->    NAME(CHARS=FNTYPE(PARG))
                  NUMBER(NUM=NOPTS(PARG))

(285)      TIMARG(¬SUP)    -->    TNUMBER(NUM(TIMARG),XVSW=XVSW(TIMARG))
(286)      TIMARG(SUP.EQ.SUP(TIMUNIT(MEM)))    -->
                  TNUMBER(NUM=NUM(TIMARG)/NUM(TIMUNIT(MEM)),XVSW=XVSW(TIMARG))
(287)      TIMARG(SUP.EQ.UNITS(SUP(TIMUNIT(MEM))))    -->
                  TIMARG(NUM=NUM(TIMARG)/NUM(SUP(TIMUNIT(MEM))),
                        SUP(TIMUNIT(MEM)))
(288)      TIMARG    -->    TIMARG(NUM=NUM*NUM(SUP),SUP=UNITS(SUP))
(289)      TNUMBER(XVSW)     -->     XVECTOR(RIGHT=NUM(TNUMBER))
(290)      TNUMBER    -->    NUMBER(NUM(TNUMBER))

(291)      COMMAS(NCOM.GT.COMCNT(MEM))     -->
                  ,    COMMAS(COMCNT(MEM)=COMCNT(MEM)+1)
(292)      COMMAS     -->    NULL
(293)      NEWLINE1    -->    OUTPUT(@11=1,@12=1)
(294)      NEWLINE2    -->    OUTPUT(@11=1,@12=2)
(295)      NEWLINE8    -->    OUTPUT(@11=1,@12=8)
(296)      COLUMN7     -->    OUTPUT(@12=7)
(297)      COLUMN8     -->    OUTPUT(@12=8)
(298)      COLUMN13    -->    OUTPUT(@12=13)
(299)      COLUMN19    -->    OUTPUT(@12=19)
```

:END OF FILE:

APPENDIX E

Encoding Rules

for

Massaging the IPD

SEMOLOGY FOR ENCODING - MASSAGING THE IPD:

```
(301)      MASSAGER    -->
                     INITIALIZE
                     RECLIST(%'ACTNLIST',LIST='ACTNLIST',LC=11)
                     RECLIST(%'MOBLIST',LIST='MOBLIST',LC=11)
                     RECLIST(%'STALIST',LIST='STALIST',LC=11)
                     RECLIST(%'DSTRLIST',LIST='DSTRLIST',LC=11)
                     RECLIST(%'SCSRLIST',LIST='SCSRLIST',LC=11)
                     RECLIST(%'MISCLIST',LIST='MISCLIST',LC=11)
                     RECLIST(%'UNITLIST',LIST='UNITLIST',LC=21)

(302)      INITIALIZE    -->
                     NULL(-EXPUSED(MEM),-NORMUSED(MEM),-QUESTSW(MEM),
                          -SUPSET(MEM),-ATTRIB(MEM),-CENTY(MEM),-PRED(MEM),
                          -SUCC(MEM),-CONDITN(MEM),-RNNO(MEM),MVARID(MEM)=1,
                          MFNID(MEM)=3,-NOREST(MEM))

(303)      RECLIST(¬CHECKED,LC.LE.LASTREC)    -->
                     REC(%@LC(RECLIST)(RECLIST),LIST(RECLIST),LC(RECLIST),
                          ENTY=@LC(RECLIST)(RECLIST),-@3(ENTY),-@4(ENTY))
                     RECLIST(LC=LC+1)
(304)      RECLIST   -->    NULL

(305)      REC(LIST.EQ.'ACTNLIST')    -->    ACTNREC(%REC)
(306)      REC(@0(ENTY).EQ.3)    -->    ERASE(%REC)
(307)      REC(CHECKED)    -->    NULL
(308)      REC(LIST.EQ.'MOBLIST')     -->    MOBREC(%REC)
(309)      REC(LIST.EQ.'STALIST')     -->    STAREC(%REC)
(310)      REC(LIST.EQ.'DSTRLIST')    -->    DSTRREC(%REC)
(311)      REC(LIST.EQ.'SCSRLIST')    -->    SCSRREC(%REC)
(312)      REC(LIST.EQ.'MISCLIST')    -->    MISCREC(%REC)
(313)      REC(LIST.EQ.'UNITLIST')    -->    UNITREC(%REC)

(314)      ACTNREC('ARRIV',¬ASNDISTR,AGENT,
                  AGENT.EQ.STRUCENTY(ASNDISTR(MEM)))    -->
                     ACTNREC(ASNDISTR=ASNDISTR(MEM),ASNDISTR(ENTY)=ASNDISTR,
                          -CHECKED(ENTY))
(315)      ACTNREC(CHECKED)    -->    NULL
(316)      ACTNREC(TIME,TIME¬$'ABSCLOCK',$'ACTIVITY')    -->
                     ACTNREC(DURATION=TIME,DURATION(ENTY)=TIME,
                          -TIME,-TIME(ENTY))
(317)      ACTNREC(TIME,TIME¬$'RELTIME',$'EVENT')    -->
                     ACTNREC(IETM=TIME,IETM(ENTY)=TIME,-TIME,-TIME(ENTY))
(318)      ACTNREC(CONDITN)    -->    ACTNREC(-CONDITN,-CONDITN(ENTY))
(319)      ACTNREC(INDIC)    -->    ACTNREC(-INDIC,-INDIC(ENTY),
                                      REACHED(ENTY)=REACHED(ACTNREC))
(320)      ACTNREC    -->    NULL

(321)      MOBREC(¬CONSUMP)    -->
                     MOBREC(CONSUMP='ONE',CONSUMP(ENTY)='ONE')
(322)      MOBREC(¬LOWINT)    -->    MOBREC(LOWINT=1,LOWINT(ENTY)=1,
                                      INTWIDTH(ENTY)=1,NUMINT(ENTY)=2)
(323)      MOBREC(INDIC)    -->    MOBREC(-INDIC,-INDIC(ENTY))
(324)      MOBREC    -->    NAMER(%MOBREC)

(325)      STAREC(SUP(QUANTITY).EQ.'UNIT')    -->
                     STAREC(QUANTITY=NUM(QUANTITY),QUANTITY(ENTY)=QUANTITY)
(326)      STAREC(¬QUANTITY($STRUC(SEG)))    -->
                     STAREC(QUANTITY=1,QUANTITY(ENTY)=1)
(327)      STAREC(¬CAPACITY($STRUC(SEG)))    -->
                     STAREC(CAPACITY='ONE',CAPACITY(ENTY)='ONE')
(328)      STAREC(¬STORIND,QUANTITY.GT.1|NUM(CAPACITY).GT.1)    -->
                     STAREC(STORIND=1,STORIND(ENTY)=1)
(329)      STAREC(INDIC)    -->    STAREC(-INDIC,-INDIC(ENTY))
(330)      STAREC    -->    NAMER(%STAREC)

(331)      DSTRREC($'FNCTN')    -->
                     NULL(IDNO(ENTY(DSTRREC))=MFNID(MEM),
                          MFNID(MEM)=MFNID(MEM)+1)
(332)      DSTRREC('NORMAL')    -->
                     NULL(IDNO(ENTY(DSTRREC))=MVARID(MEM),
                          MVARID(MEM)=MVARID(MEM)+1,NORMUSED(MEM))
(333)      DSTRREC('EXPON')    -->    NULL(EXPUSED(MEM))
(334)      DSTRREC    -->    MCHECKED(%DSTRREC)

(335)      SCSRREC($'SUCDSCR1')    -->
                     NULL(IDNO(ENTY(SCSRREC))=MFNID(MEM),
                          MFNID(MEM)=MFNID(MEM)+1,
                          FNARG(ENTY(SCSRREC))=SUCARG(SCSRREC),
                          DORC(ENTY(SCSRREC))="D")
(336)      SCSRREC    -->    MCHECKED(%SCSRREC)

(337)      MISCREC(INDIC,¬$'ENTITY')    -->    MISCREC(-INDIC,-INDIC(ENTY))
(338)      MISCREC    -->    MCHECKED(%MISCREC)

(339)      UNITREC    -->    MCHECKED(%UNITREC)


(340)      NAMER(IDNAME(SUP))    -->    NAMER2(%NAMER,IDNAME(SUP))
(341)      NAMER    -->    NAMER2(%NAMER,IDNAME=NAME(SUP))

(342)      NAMER2    -->    NULL(%NAMER2,@8=IDNAME,@9=IDNO,FORMSYMBOL,
```

```
                    IDNAME(ENTY)=@8)
(343)    MCHECKED   -->   NULL
(344)    ERASE   -->   NULL(@LC(ERASE)(LIST(ERASE))='NULL')
(345)    NULL   -->   OUTPUT

:END OF FILE:
```

APPENDIX F

Encoding Rules

for

Asking Questions

(The Interrogator)

SEMOLOGY FOR ENCODING QUESTIONS:

```
(401)     INTERROGATOR   -->
                    SETAGL(%ACPTR(MEM),LC=11)
                    RECLISTI(%'ACTNLIST',LIST='ACTNLIST',LC=11)
                    RECLISTI(%'ACTNLIST',LIST='ACTNLIST',LC=11,K=3)
                    RECLISTI(%'MOBLIST',LIST='MOBLIST',LC=11)
                    RECLISTI(%'STALIST',LIST='STALIST',LC=11)
                    MEMREC1
(402)     RECLISTI(LC.LE.LASTREC,¬QUESTSW(MEM))   -->
                    RECI(%@LC(RECLISTI)(RECLISTI),K(RECLISTI),LIST(RECLISTI),
                         CENTY=@LC(RECLISTI)(RECLISTI))
                    RECLISTI(LC=LC+1)
(403)     RECLISTI   -->   NULL
(404)     RECI(LIST.EQ.'ACTNLIST',K.EQ.3)   -->   ACTNREC3(%RECI)
(405)     RECI(CHECKED)   -->   NULL
(406)     RECI(LIST.EQ.'ACTNLIST')   -->   ACTNREC1(%RECI)
(407)     RECI(LIST.EQ.'MOBLIST')   -->   MOBREC1(%RECI)
(408)     RECI(LIST.EQ.'STALIST')   -->   STAREC1(%RECI)
(409)     ACTNREC1(¬MOBENATR(CENTY))   -->
                    INTSENT1(%ACTNREC1,SUBJECT='WHATREC',SING,
                              MOBENATR=AGORGL($(SEG)),ATTRIB=MOBENATR,
                              SUPSET(MEM)='MOBENTY',QUESMK,
                              ¬MOBENATR,MOBENATR='AGENT',ATTRIB='AGENT')
(410)     ACTNREC1(LIMITCHK)   -->   NULL
(411)     ACTNREC1(¬LOCATION(CENTY))   -->
                    INTSENT1(%ACTNREC1,ATTRIB='LOCATION',
                              SUPSET(MEM)='LOCDESCR')
(412)     ACTNREC1(¬IETM(CENTY),'ARRIV'|'ENTER')   -->
                    INTSENT1(%ACTNREC1,ATTRIB='IETM')
(413)     ACTNREC1(IETM)   -->
                    QUANREC(%IETM(ACTNREC1),CENTY(ACTNREC1),AENTY=CENTY,
                             ATTRIB='IETM',VALSET='ABSTIME',ATTRIB1='IETM')
                    ACTNREC2(%ACTNREC1,-IETM)
(414)     ACTNREC1(¬DURATION(CENTY),$'ACTIVITY')   -->
                    INTSENT1(%ACTNREC1,ATTRIB='DURATION')
(415)     ACTNREC1(DURATION$'COND',CONDENTY(DURATION)¬$'STATENTY')   -->
                    INTSENT1(%CENTY(ACTNREC1),CENTY(ACTNREC1),ATTRIB='DURATION',
                              CAN,NEG,REASON=CHARS('REASON1'))
                    INTSENT3(%CENTY(ACTNREC1),ATTRIB='DURATION')
(416)     ACTNREC1(DURATION,DURATION¬$'COND')   -->
                    QUANREC(%DURATION(ACTNREC1),CENTY(ACTNREC1),AENTY=CENTY,
                             ATTRIB='DURATION',VALSET='ABSTIME',
                             ATTRIB1='DURATION')
                    ACTNREC2(%ACTNREC1,-DURATION)
(417)     ACTNREC1(ASNDISTR)   -->
                    REC1(%ASNDISTR(ACTNREC1),CENTY(ACTNREC1),AENTY=CENTY,
                         XYC=101)
                    ACTNREC2(%ACTNREC1,-ASNDISTR)
(418)     ACTNREC1(¬SUCC(CENTY),¬'LEAV')   -->
                    INTSENT2('DO',PRED=ACTNREC1,-IETM(PRED),
                              ATTRIB(MEM)='SUCC',PRED(MEM)=CENTY(ACTNREC1),
                              CENTY(MEM)=CENTY(ACTNREC1),AENTY=CENTY(ACTNREC1))
(419)     ACTNREC1(SUCC)   -->
                    SCSRREC1(%SUCC(ACTNREC1),CENTY(ACTNREC1),AENTY=CENTY,
                              ENTY=SUCC(ACTNREC1),ATTRIB1='SUCC')
                    ACTNREC2(%ACTNREC1,-SUCC)
(420)     ACTNREC1   -->   ICHECKED(%ACTNREC1)
(421)     ACTNREC2(¬QUESTSW(MEM))   -->   ACTNREC1(%ACTNREC2)
(422)     ACTNREC2   -->   NULL
(423)     ACTNREC3(¬REACHED,¬'ARRIV',¬'ENTER')   -->
                    INTSENT2('DO',SUCC=ACTNREC3,-IETM(SUCC),
                              ATTRIB(MEM)='PRED',SUCC(MEM)=CENTY(ACTNREC3),
                              AENTY=CENTY(ACTNREC3))
(424)     ACTNREC3   -->   NULL
(425)     MOBREC1(WEIGHT(CENTY))   -->
                    QUANREC(%WEIGHT(MOBREC1),CENTY(MOBREC1),AENTY=CENTY,
                             ATTRIB='WEIGHT',VALSET='ABSWEIT',ATTRIB1='WEIGHT')
(426)     MOBREC1   -->   ICHECKED(%MOBREC1)
(427)     STAREC1   -->   ICHECKED(%STAREC1)
(428)     MEMREC1(QUESTSW(MEM))   -->   NULL
(429)     MEMREC1(¬PROBTIME(MEM))   -->
                    PHRASE(CHARS=" HOW LONG SHALL THE SIMULATION BE RUN?",
                           ATTRIB(MEM)='PROBTIME',CENTY(MEM)=MEM,
                           SUPSET(MEM)='ABSTIME',QUESTSW(MEM))
(430)     MEMREC1(¬TIMUNIT(MEM))   -->
                    PHRASE(CHARS=" WHAT IS THE BASIC TIME UNIT TO BE USED")
                    PHRASE(CHARS=" IN THE MODEL?",
                           ATTRIB(MEM)='TIMUNIT',CENTY(MEM)=MEM,
                           SUPSET(MEM)='ABSTIME',QUESTSW(MEM))
(431)     MEMREC1   -->   NULL
(432)     ICHECKED   -->   NULL
(433)     QUANREC($VALSET)   -->   NULL
(434)     QUANREC($'STDIST')   -->
                    DSTRREC1(%QUANREC,CENTY=@ATTRIB1(SEG)(CENTY))
(435)     QUANREC('TYPTABL',ATTRIB.EQ.'IETM')   -->
                    INTSENT1(%AENTY(QUANREC),AENTY(QUANREC),CENTY(QUANREC),
                              ATTRIB(QUANREC),CAN,NEG,REASON=CHARS('REASON3'))
                    INTSENT3(%CENTY(QUANREC),ATTRIB=ATTRIB1(QUANREC))
(436)     QUANREC('TYPTABL')   -->   REC1(%QUANREC,XYC=101)
(437)     QUANREC   -->
                    INTSENT1(%AENTY(QUANREC),AENTY(QUANREC),ATTRIB(QUANREC),
                              CENTY(QUANREC),CAN,NEG,REASON=CHARS('REASON2'))
```

```
                INTSENT3(%CENTY(QUANREC),ATTRIB=ATTRIB1(QUANREC))
(438)     DSTRREC1(¬MEAN)    -->
                DSTRREC3(%DSTRREC1)
                INTSENT3(%DSTRREC1,ATTRIB='MEAN')
(439)     DSTRREC1(MEAN)    -->
                QUANREC(%MEAN(DSTRREC1),AENTY(DSTRREC1),ATTRIB(DSTRREC1),
                     VALSET(DSTRREC1),ATTRIB1='MEAN',
                     CENTY(DSTRREC1))
                DSTRREC2(%DSTRREC1)
(440)     DSTRREC2(QUESTSW(MEM))    -->    NULL
(441)     DSTRREC2('NORMAL',¬STDEV)    -->
                DSTRREC3(%DSTRREC2)
                INTSENT3(%DSTRREC2,ATTRIB='STDEV')
(442)     DSTRREC2(STDEV)    -->
                QUANREC(%STDEV(DSTRREC2),AENTY(DSTRREC2),ATTRIB(DSTRREC2),
                     VALSET(DSTRREC2),ATTRIB1='STDEV',
                     CENTY(DSTRREC2))
(443)     DSTRREC2('UNIFORM',¬RANGE)    -->
                DSTRREC3(%DSTRREC2)
                INTSENT3(%DSTRREC2,ATTRIB='RANGE')
(444)     DSTRREC2(RANGE)    -->
                QUANREC(%RANGE(DSTRREC2),AENTY(DSTRREC2),ATTRIB(DSTRREC2),
                     VALSET(DSTRREC2),ATTRIB1='RANGE',
                     CENTY(DSTRREC2))
(445)     DSTRREC2    -->    NULL
(446)     DSTRREC3(¬QAMODE(MEM))    -->
                INTSENT1(%AENTY(DSTRREC3),ATTRIB(DSTRREC3),CENTY(DSTRREC3))
(447)     DSTRREC3    -->    NULL(QUESTSW(MEM),ATTRIB(MEM)=ATTRIB(DSTRREC3),
                          CENTY(MEM)=CENTY(DSTRREC3))
(448)     SCSRREC1(QUESTSW(MEM))    -->    NULL
(449)     SCSRREC1(OPENACT)    -->
                SCSRREC1(%OPENACT(SCSRREC1),AENTY(SCSRREC1),
                     CENTY=@ATTRIB1(SCSRREC1)(CENTY(SCSRREC1)),
                     ATTRIB1='OPENACT')
                SCSRREC1(-OPENACT)
(450)     SCSRREC1(CLOSACT)    -->
                SCSRREC1(%CLOSACT(SCSRREC1),AENTY(SCSRREC1),
                     CENTY=@ATTRIB1(SCSRREC1)(CENTY(SCSRREC1)),
                     ATTRIB1='CLOSACT')
                SCSRREC1(-CLOSACT)
(451)     SCSRREC1('QTYP',¬SUCARG)    -->
                SCSRREC2(%SCSRREC1)
                SENT('BE',QUESTSW(MEM),INTRGPH=" WHERE",SUBJECT='LINEREC',
                     SUPSET(MEM)='LOCDESCR',ATTRIB(MEM)='SUCARG',
                     CENTY(MEM)=@ATTRIB1(SCSRREC1)(CENTY(SCSRREC1)))
(452)     SCSRREC1($'SUCDSCR2',SUCARG¬$'STATENTY')    -->
                PHRASE(CHARS=" THE ENTITY IN THE FOLLOWING CONDITION")
                PHRASE(CHARS=" MUST BE A STATIONARY ENTITY:  ")
                SUCCERR(%SCSRREC1)
(453)     SCSRREC1($'SUCDSCR2',ACT2)    -->
                SCSRREC2(%SCSRREC1)
                INTSENT2('DO',AENTY(SCSRREC1),CONDITN='OTHERWIS',
                     ATTRIB(MEM)=ACT2(SCSRREC1),
                     CENTY(MEM)=@ATTRIB1(SCSRREC1)(CENTY(SCSRREC1)))
(454)     SCSRREC1(XYLAST)    -->    REC1(%SCSRREC1,XYC=101)
(455)     SCSRREC1($'ACTION',¬CHECKED)    -->
                ACTNREC1(%SCSRREC1,CENTY=@ATTRIB1(SEG)(CENTY(SEG)),
                     AENTY=CENTY,REACHED(AENTY),LIMITCHK)
(456)     SCSRREC1($'ACTION')    -->
                NULL(REACHED(@ATTRIB1(SCSRREC1)(CENTY(SCSRREC1))))
(457)     SCSRREC1    -->    NULL
(458)     SCSRREC2(¬QAMODE(MEM))    -->
                SUCCDESC(%@ATTRIB1(SCSRREC2)(CENTY(SCSRREC2)),
                     PRED=%AENTY(SCSRREC2),-IETM(PRED),
                     OPENACT.NE.0,-CLOSACT)
(459)     SCSRREC2    -->    NULL
(460)     REC1(QUESTSW(MEM))    -->    NULL
(461)     REC1(XYC.GT.XYLAST,'TYPTABL')    -->
                REC2(%REC1,CENTY=@ATTRIB1(SEG)(CENTY),XYC=102)
(462)     REC1(XYC.GT.XYLAST)    -->    REC3(%REC1,XYC=XYLAST-1)
(463)     REC1(¬@XYC,'TYPTABL')    -->
                PHRASE(CHARS=" THERE IS SOMETHING MISSING IN THE")
                PHRASE(CHARS=" FOLLOWING STATEMENT:  ")
                INTSENT1(%AENTY(REC1),AENTY(REC1),ATTRIB(REC1),
                     CENTY(REC1))
                INTSENT3(%AENTY(REC1),ATTRIB(REC1))
(464)     REC1(¬@XYC)    -->
                PHRASE(CHARS=" THERE IS SOMETHING MISSING IN THE")
                PHRASE(CHARS=" FOLLOWING STATEMENT:  ")
                REC4(%REC1)
(465)     REC1(@XYC$'ACTION',¬CHECKED(@XYC))    -->
                ACTNREC1(%@XYC(REC1)(REC1),CENTY=@XYC(REC1)(REC1),
                     LIMITCHK)
                REC1(XYC=XYC+1)
(466)     REC1    -->    REC1(XYC=XYC+1,@XYC$'ACTION',REACHED(@XYC))
(467)     REC2(XYC.GT.XYLAST|QUESTSW(MEM))    -->    NULL
(468)     REC2    -->
                QUANREC(%@XYC(REC2)(REC2),AENTY(REC2),ATTRIB(REC2),
                     VALSET(REC2),ATTRIB1=XYC(REC2),CENTY(REC2))
                REC2(XYC=XYC+2)
(469)     REC3(SUP(@XYC).NE.'DECIMAL'|NUM(@XYC).GT.990,NUM(@XYC).LT.1010)    -->
                NULL
(470)     REC3    -->
                PHRASE(CHARS=" THE FOLLOWING PERCENTAGES DO NOT TOTAL 100:",
                       NOREST(MEM))    #    #
```

```
                  REC4(%REC3)
(471)      REC4('TYPDIST')    -->    ASNERR(%REC4)
(472)      REC4    -->    SUCCERR(%REC4)
(473)      ASNERR     -->    ASNDESC(%ASNDISTR(AENTY(ASNERR)))
                  PHRASE(CHARS=" PLEASE RESPECIFY.",QUESTSW(MEM),
                       LASTME(MEM)=STRUCENTY(ASNERR),
                       -ASNDISTR(CENTY(ASNERR)))
(474)      SUCCERR    -->    SUCCDESC(%SUCC(AENTY(SUCCERR)),PRED=%AENTY(SUCCERR),
                                 -IETM(PRED),OPENACT.NE.0,-CLOSACT)
                  ACTNREC1(%AENTY(SUCCERR),CENTY=AENTY(SUCCERR),-SUCC,
                         -SUCC(CENTY))
(475)      INTSENT1($'MOBENTY')    -->
                  SENT(%INTSENT1,QUESTSW(MEM),LASTME(MEM)=AENTY,-LASTSE(MEM),
                      -LASTLD(MEM),CENTY(MEM)=CENTY,ATTRIB(MEM)=ATTRIB)
(476)      INTSENT1    -->
                  SENT(%INTSENT1,-PRED,-SUCC,-CONDITN,QUESTSW(MEM),
                      LASTME(MEM)=@MOBENATR,LASTSE(MEM)=LOCOBJ(LOCATION),
                      LASTLD(MEM)=SUP(LOCATION),CENTY(MEM)=CENTY,
                      ATTRIB(MEM)=ATTRIB,ATTRIB.EQ.MOBENATR,-ATTRIB)
(477)      INTSENT2    -->
                  SENT(%INTSENT2,INTRGPH=" WHAT",
                      SUBJECT=@MOBENATR(AENTY(SEG))(AENTY),
                      QUESTSW(MEM),LASTME(MEM)=SUBJECT,
                      LASTSE(MEM)=LOCOBJ(LOCATION),LASTLD(MEM)=SUP(LOCATION))
(478)      INTSENT3    -->
                  SENT(%INTSENT3,SUPSET(MEM)='VALU',-@ATTRIB,
                      ATTRIB(MEM)=ATTRIB)

:END OF FILE:
```

APPENDIX G

Encoding Rules

for

Answering Questions

SEMOLOGY FOR ENCODING ANSWERS TO QUESTIONS:

```
(501)     QUESTION(¬ENTY(@11))   -->   QUEST3(%QUESTION)
(502)     QUESTION   -->   QUEST1(%ENTY(@11(QUESTION)),ATTRIB(@11(QUESTION)),
                              VAL(@11(QUESTION)),
                              VALQUAN=QUANTITY(@11(QUESTION)))
(503)     QUEST1(INDIC)   -->   QUEST1(-INDIC)
(504)     QUEST1(¬MOBENATR,GOAL$'MOBENTY')   -->   QUEST1(MOBENATR='GOAL')
(505)     QUEST1(¬MOBENATR,AGENT$'MOBENTY')   -->   QUEST1(MOBENATR='AGENT')
(506)     QUEST1(TIME,$'ACTIVITY')   -->   QUEST1(DURATION=TIME,-TIME)
(507)     QUEST1(TIME,$'EVENT')   -->   QUEST1(IETM=TIME,-TIME)
(508)     QUEST1(ATTRIB.EQ.'TIME',$'ACTIVITY')   -->
                 QUEST1(ATTRIB='DURATION')
(509)     QUEST1(ATTRIB.EQ.'TIME',$'EVENT')   -->   QUEST1(ATTRIB='IETM')
(510)     QUEST1   -->   QUEST2(%QUEST1)
(511)     QUEST2(¬ATTRIB)   -->   ANSWR('YES')
                 SENT(%QUEST2,-PRED,-SUCC)
(512)     QUEST2(¬@ATTRIB)   -->   ANSWR('IDK')
(513)     QUEST2(¬VAL|VAL$'INTRGVAL')   -->   QUEST5(%QUEST2)
(514)     QUEST2(@ATTRIB.NE.ENTY(VAL))   -->   ANSWR('NO')   QUEST5(%QUEST2)
(515)     QUEST2(VALQUAN,¬QUANTITY)   -->   ANSWR('IDK')
(516)     QUEST2(VALQUAN,VALQUAN.NE.QUANTITY)   -->
                 ANSWR('NO')
                 SENT(%QUEST2,ATTRIB='QUANTITY')
(517)     QUEST2   -->   ANSWR('YES')   QUEST5(%QUEST2)
(518)     QUEST3(SUP(@11).NE.'DO')   -->   ANSWR('IDK')
(519)     QUEST3(PRED)   -->   QUEST4(%ENTY(PRED(QUEST3)))
(520)     QUEST3(PRED(MEM))   -->   QUEST4(%PRED(MEM))
(521)     QUEST3   -->   ANSWR('PBMS')
(522)     QUEST4(SUCC)   -->   SUCCDESC(%SUCC(QUEST4),PRED=QUEST4,-IETM(PRED))
(523)     QUEST4   -->   ANSWR('IDK')
(524)     QUEST5(ATTRIB.EQ.'LOCATION',$'ACTION')   -->   QUEST6(%QUEST5)
(525)     QUEST5(ATTRIB.EQ.'IETM',IETM$'ABSTIME')   -->
                 QUEST6(%QUEST5)
(526)     QUEST5(ATTRIB.EQ.'DURATION',DURATION$'ABSTIME')   -->
                 QUEST6(%QUEST5)
(527)     QUEST5   -->   SENT(%QUEST5,-PRED,-SUCC)
(528)     QUEST6   -->   SENT(%QUEST6,-ATTRIB,-PRED,-SUCC)
(529)     ANSWR('YES')   -->   PHRASE(CHARS=" YES,")
(530)     ANSWR('NO')   -->   PHRASE(CHARS=" NO,")
(531)     ANSWR('IDK')   -->   PHRASE( CHARS=" I DON'T KNOW.")
(532)     ANSWR('PBMS')   -->   PHRASE(CHARS=" PLEASE BE MORE SPECIFIC.")
```

:END OF FILE:

A P P E N D I X H

Encoding Rules

for

the Linkage from Decoding

SEMOLOGY FOR ENCODING - THE LINKAGE FROM DECODING:

```
(551)      ENCODING(ETYPE.EQ.'QUESTION')    -->   QUESTION(%ENCODING)
(552)      ENCODING(ETYPE.EQ.'REPLYERR')    -->
                    PHRASE(CHARS=" THAT IS NOT A REASONABLE REPLY.  TRY AGAIN.")
(553)      ENCODING(ETYPE(MEM).EQ.'ENGLISH')    -->
                    MASSAGER   INTERROGATOR   ENGLISH
(554)      ENCODING(ETYPE(MEM).EQ.'GPSSPROG')    -->
                    MASSAGER   INTERROGATOR   GPSSPROG
(555)      ENCODING(ETYPE(MEM).EQ.'QUESANSW')    -->
                    MASSAGER(QAMODE(MEM))   INTERROGATOR   COMPMSG
(556)      ENCODING(ETYPE(MEM).EQ.'CHECKIPR')    -->
                    MASSAGER(-QAMODE(MEM))   INTERROGATOR   COMPMSG
(557)      ENCODING   -->   NULL

(558)      COMPMSG(¬QUESTSW(MEM))   -->
                    NEWPAR
                    PHRASE(CHARS=" THE PROBLEM STATEMENT IS COMPLETE.",
                           -QAMODE(MEM))
(559)      COMPMSG   -->   NULL
```

:END OF FILE:

APPENDIX I

Decoding Rules

for

English

MORPHOLOGY FOR DECODING ENGLISH:

```
(601)    #   DIGIT   -->   NUMBERP
(602)    NUMBERP   DIGIT   -->   NUMBERP

(603)    #   LETTER   -->   CHARSTRING(LOOKUP)
(604)    CHARSTRING   LETTER   -->   CHARSTRING(LOOKUP)
(605)    CHARSTRING(SUP)   -->   STEM*1(SUP(CHARSTRING),PS($(SEG)))

(606)    STEM(PS.EQ.'NOUNS')   -->   NOUNS(SUP(STEM),SUP(SUP).EQ.'DECIMAL',
                                        NUM(SUP),DECENTY=SUP,'DECIMAL',
                                        ENTY=LASTME(MEM))
(607)    STEM(PS.EQ.'VERBS')   -->   VERBS(SUP(STEM),SUFX*)
(608)    STEM(PS.EQ.'NOUN')   -->   NOUNP(%SUP(STEM),-PS,-NAME)
(609)    STEM(PS.EQ.'VERB')   -->   VERBP(%SUP(STEM),-PS,-NAME)
(610)    STEM(PS.EQ.'ADJ')   -->   ADJP(SUP(STEM),ENTY=SUP,
                                     SUP(SUP).EQ.'UNIT',NUM(SUP),'UNIT')
(611)    STEM(PS.EQ.'ADV')   -->   ADVP(SUP(STEM))
(612)    STEM(PS.EQ.'CONJ')   -->   CONJP(SUP(STEM))
(613)    STEM(PS.EQ.'PREP')   -->   PREPP(SUP(STEM))
(614)    STEM(PS.EQ.'PRON')   -->   PRONP(SUP(STEM),PRONIND*)

(615)    NOUNS('HALF')   -   R   A   N   G   E   -->   NOUNS('RANGE')
(616)    NOUNS('DEVIATIO')   N   -->   NOUNS('STDEV')
(617)    NOUNS('SIMULATI')   O   N   -->   NOUNS

(618)    NOUNS   -->   NOUNP(%NOUNS,-INDIC,SING)
(619)    NOUNS(¬ES*)   S   -->   NOUNP(%NOUNS,-INDIC,PLUR)
(620)    NOUNS(ES*)   E   S   -->   NOUNP(%NOUNS,-INDIC,PLUR)

(621)    NOUNP   '   -->   NOUNP(POSS)
(622)    NOUNP(POSS)   S   -->   NOUNP

(623)    VERBS(N)   N   -->   VERBS(-SUFX,ING,ED,ER)
(624)    VERBS(R)   R   -->   VERBS(-SUFX,ING,ED,ER)
(625)    VERBS(E)   E   -->   VERBS(-SUFX,NSFX)
(626)    VERBS(ES)   E   -->   VERBS(-SUFX,S)

(627)    VERBS(NSFX)   -->   VERBP(SUP(VERBS),PLUR,INFF)
(628)    VERBS(S)   S   -->   VERBP(SUP(VERBS),SING)
(629)    VERBS(ED)   E   D   -->   VERBP(SUP(VERBS),PASTPART,PASTF)
(630)    VERBS(EN)   E   N   -->   VERBP(SUP(VERBS),PASTPART)
(631)    VERBS(ING)   I   N   G   -->   VERBP(SUP(VERBS),PRESPART)
(632)    VERBS('ARRIV',E)   A   L   -->   VERBP('ARRIV',PRESPART)
(633)    VERBS('BE')   -->   VERBP('BE',INF)

(634)    VERBP(PRESPART)   S   -->   VERBP(S)
(635)    VERBP(FINITE)   N   '   T   -->   VERBP(NEG)
(636)    VERBP('HAV',INFF)   -->   VERBP(-PLUR,-INFF,INF)

(637)    ADJP('DISTRIBU')   T   E   D   -->   ADJP
(638)    ADJP('EXPON')   E   N   T   I   A   L   -->   ADJP
(639)    ADJP('AVAILABL')   E   -->   ADJP
(640)    ADJP('UNAVAIL')   A   B   L   E   -->   ADJP
(641)    ADJP('IMMEDIAT')   E   -->   ADJP

(642)    ADJP(LY*)   L   Y   -->   ADVP(%ADJP)

(643)    ADVP('OTHERWIS')   E   -->   ADVP
```

LEXOLOGY FOR DECODING ENGLISH:

```
(644)    NOUNP(ENDOFWORD)   -->   NOUN(%NOUNP,WRECOGNIZED(MEM))
(645)    VERBP(ENDOFWORD)   -->   VERB(%VERBP,WRECOGNIZED(MEM))
(646)    ADJP(ENDOFWORD)   -->   ADJ(%ADJP,WRECOGNIZED(MEM))
(647)    ADVP(ENDOFWORD)   -->   ADV(%ADVP,WRECOGNIZED(MEM))
(648)    CONJP(ENDOFWORD)   -->   CONJ(%CONJP,WRECOGNIZED(MEM))
(649)    PREPP(ENDOFWORD)   -->   PREP(%PREPP,WRECOGNIZED(MEM))
(650)    PRONP(ENDOFWORD)   -->   PRON(%PRONP,WRECOGNIZED(MEM))
(651)    NUMBERP(ENDOFWORD)   -->   NUMBER(CONVERT,NUM=@1,'UNIT')

(652)    CHARSTRING(ENDOFWORD)   /PUNCB   -->   KWORD(%CHARSTRING),
                   WORDTEST(WRECOGNIZED=WRECOGNIZED(MEM),-WRECOGNIZED(MEM))

(653)    WORDTEST(¬WRECOGNIZED)   -->   ERROR

(654)    NUMBER   -->   SRCHREC(%NUMBER,LIST='UNITLIST',LC=10,TYPE='ADJPH')

(655)    NOUN   -->   NOUN1*1(%NOUN)

(656)    NOUN1($'MOBENTY')   -->   NOUN2*1(%NOUN1,LIST='MOBLIST',LC=11,
                                     ENTY=@11(LIST),LASTE='LASTME')
(657)    NOUN1($'STATENTY')   -->   NOUN2(%NOUN1,LIST='STALIST',LC=11,
                                    ENTY=@11(LIST),LASTE='LASTSE')
(658)    NOUN1($'ENTITY')   -->   SRCHREC(%NOUN1,LIST='MISCLIST',LC=10,
                                    TYPE='NOUNPH')
(659)    NOUN1(SUP(SUP).NE.'DECIMAL')   -->   NOUNPH(%NOUN1)

(660)    NOUN2(LC.GT.LASTREC(LIST))   -->
                   RECORD(%NOUN2,LASTREC(LIST)=LC,IDNO=LC-10,
```

```
                  @LASTREC(LIST)(LIST)=RECORD,-LIST,-LC,-ENTY,-LASTE),
          NOUNPH(%NOUN2,ENTY=@LASTREC(LIST)(LIST),
                  @LASTE(SEG)(MEM)=ENTY,-LASTE)
(661)   NOUN2(SUP.EQ.SUP(ENTY))   -->
          NOUNPH(%NOJN2,@LASTE(SEG)(MEM)=ENTY,-LASTE)
(662)   NOUN2   -->   NOUN2(LC=LC+1,ENTY=@LC(SEG)(LIST))

(663)   PRON(PERS1,LASTME(MEM)$'PERSON')   -->
          NOUNPH(%PRON,PRM,ENTY=LASTME(MEM))
(664)   PRON(PERS2,LASTME(MEM),LASTME(MEM)¬$'PERSON')   -->
          NOUNPH(%PRON,PRM,ENTY=LASTME(MEM))
(665)   PRON(PERS2,LASTSE(MEM))   -->
          NOUNPH(%PRON,PRM,ENTY=LASTSE(MEM))


(666)   NOUN('PROBLEM'|'SIMULATI')   NOUNPH('TIME',¬OBJ)   -->
          NOUNPH('PROBTIME',OBJ=%NOUN,ENTY(OBJ)=MEM)

(667)   NOUNPH($'ATTR',OBJ,¬ENTY,ENTY=@SUP(SEG)(ENTY(OBJ)),
               'TIME',¬ENTY,ENTY=DURATION(ENTY(OBJ)),
               ¬ENTY,ENTY=IETM(ENTY(OBJ)),1.NE.1)          -->   NULL
(668)   NOUNPH(¬PRM,¬LOCATION,$'STATENTY'|'LINE')
                    ADVIAL(ENTY,ATTRIB.EQ.'LOCATION',
                           ¬LOCATION(ENTY(NOUNPH))|
                           LOCATION(ENTY(NOUNPH)).EQ.ENTY)   -->
          NOUNPH(LOCATION=ENTY(ADVIAL),LOCATION(ENTY)=LOCATION)
(669)   NOUNPH(¬PRM,$'ATTR',¬OBJ)   VERBPH(TOINF)   -->
          NOUNPH(OBJ=VERBPH)
(670)   NOUNPH(¬PRM,$'ATTR',¬OBJ)   PREPPH('OF'|'FOR',NOUNAL(OBJ))   -->
          NOUNPH(OBJ(PREPPH))
(671)   NOUNPH(¬PRM,'TIME',¬OBJ)   PREPPH('BETWEEN',NOUNAL(OBJ))   -->
          NOUNPH('IETM',OBJ(PREPPH))
(672)   NOUNPH(POSS)   NOUNPH(¬DETERM,$'ATTR',¬OBJ)   -->
          NOUNPH(%NOUNPH#2,OBJ=NOUNPH#1)
(673)   NOUNPH(¬PRM,'DECIMAL')   PREPPH('OF',ENTY(OBJ))   -->
          NOUNPH(ENTY(OBJ(PREPPH)))
(674)   NOUNPH(NUM,$'ABSTIME',¬DETERM)   -->
          KWORD('TIMENTY',ENTY(NOUNPH))
(675)   NOUNPH(QUESTSW(MEM))   -->   REPLYPH(%NOUNPH)

(676)   NOUNPH($'QUANVAL',NUM)   PREPPH('FOR',ENTY(OBJ)$'ENTITY')   -->
          NNPHTTXY(X=ENTY(OBJ(PREPPH)),Y=ENTY(NOUNPH))

(677)   ADJPH($'VALU',¬$'STDIST')   PREPPH('FOR',ENTY(OBJ)$'ENTITY')   -->
          NNPHTTXY(X=ENTY(OBJ(PREPPH)),Y=ENTY(ADJPH))

(678)   NNPHTTXY   -->   NNPHTT(@101=X(NNPHTTXY),@102=Y(NNPHTTXY),XYLAST=102)

(679)   NNPHTT   COMMA   -->   NNPHTT
(680)   NNPHTT   NNPHTTXY   -->
          NNPHTT(XYLAST=XYLAST+1,@XYLAST=X(NNPHTTXY),
                 XYLAST=XYLAST+1,@XYLAST=Y(NNPHTTXY))
(681)   NNPHTT   AND   NNPHTTXY   -->
          RECORD(%NNPHTT,'TYPTABL',FNARG='PARAM1',DORC="D",
                 XYLAST=XYLAST+1,@XYLAST=X(NNPHTTXY),
                 XYLAST=XYLAST+1,@XYLAST=Y(NNPHTTXY),
                 LASTREC('DSTRLIST')=LASTREC('DSTRLIST')+1,
                 @LASTREC('DSTRLIST')('DSTRLIST')=RECORD),
          NOUNPH('TYPTABL',ENTY=@LASTREC('DSTRLIST')('DSTRLIST'))


(682)   ADJ($'VALU',¬$'STDIST')   -->   ADJPH(%ADJ)

(683)   ADJPH(QUESTSW(MEM))   -->   REPLYPH(%ADJPH)

(684)   ADJPH('UNIT')   NOUNPH('PERCENT')   -->
          SRCHREC('DECIMAL',NJM=10*NUM(ADJPH),LIST='MISCLIST',
                  LC=10,TYPE='NOUNPH',MENTY=LASTME(MEM))
(685)   ADJPH('UNIT')   NOUNPH('DECIMAL',NUM.NE.1000)   -->
          SRCHREC(%NOUNPH,PRM,NUM=NUM(ADJPH)*NUM,LIST='MISCLIST',
                  LC=10,TYPE='NOUNPH',MENTY=ENTY)
(686)   ADJPH('UNIT')   NOUNPH(¬DETERM,$'QUANVAL',¬'PERCENT',
                           ¬'DECIMAL',¬NUM)   -->
          SRCHREC(%NOUNPH,PRM,NUM(ADJPH),LIST='MISCLIST',
                  LC=10,TYPE='NOUNPH')
(687)   ADJPH('UNIT')   NOUNPH(¬DETERM,$'ENTITY',¬QUANTITY)   -->
          NOUNPH(PRM,QUANTITY=ADJPH)

(688)   ADJPH('UNIT')   PREP('TO')   ADJPH('UNIT')
                  NOUNPH(¬DETERM,$'QUANVAL',¬'PERCENT',¬NUM)   -->
          SRCHREC(SUP(NOUNPH),NUM=NUM(ADJPH#1)+NUM(ADJPH#2),NUM=NUM/2,
                  RANGE=NUM-NUM(ADJPH#1),LIST='MISCLIST',LC=10,
                  TYPE='NNPHUU')

(689)   NNPHUU(RANGE)   -->   SRCHREC(%NNPHUU,NUM=RANGE,MEAN=ENTY,-RANGE,
                                 -RANGE(ENTY),-ENTY,LC=10)
(690)   NNPHUU   -->
          RECORD('UNIFORM',MEAN(NNPHUU),RANGE=ENTY(NNPHUU),
                 -MEAN(RANGE),
                 LASTREC('DSTRLIST')=LASTREC('DSTRLIST')+1,
                 @LASTREC('DSTRLIST')('DSTRLIST')=RECORD),
          NOUNPH('UNIFORM',ENTY=@LASTREC('DSTRLIST')('DSTRLIST'))
```

```
(691)   ADJ('STANDARD')   NOUNPH(¬PRM,'STDEV')   -->   NOUNPH(PRM)
(692)   ADJ($'DET')   NOUNPH(¬DETERM)   -->   NOUNPH(PRM,DETERM=SUP(ADJ))
(693)   ADJ('THIS'|'THESE')   NOUNPH(NOUNAL,LR.GT.10)   -->   NOUNPH(ENTY=@11)
(694)   ADJ('SAME')   ADV('AS')   -->   SAMEPH
(695)   ADJ('THE')   SAMEPH   -->   SAMEPH
(696)   SAMEPH   NOUNPH($'ATTR',ENTY)   -->
                 NOUNPH(PRM,¬DETERM,DETERM='THE')

(697)   ADJ('LESS')   CONJ('THAN')   -->   COMPPH('LT')
(698)   ADJ('EQUAL')   PREP('TO')   -->   COMPPH('EQ')
(699)   ADJ('GREATER')   CONJ('THAN')   -->   COMPPH('GT')

(700)   COMPPH ADJPH('UNIT')   -->   ADJPH1(SUP(COMPPH),OBJREL=ENTY(ADJPH))
(701)   ADJPH1   -->   SRCHREC(%ADJPH1,LIST='MISCLIST',LC=10,TYPE='ADJPH')

(702)   ADJ('WHAT')   -->   ADV('WHAT')
(703)   ADV('HOW')   ADV('OFTEN')   -->   ADV('HOWOFTEN')
(704)   ADV('HOW')   ADV('LONG')   -->   ADV('HOWLONG')

(705)   ADJ($'STDIST')   -->
                 RECORD(SUP(ADJ),LASTREC('DSTRLIST')=LASTREC('DSTRLIST')+1,
                        @LASTREC('DSTRLIST')('DSTRLIST')=RECORD),
                 ADJPH(SUP(ADJ),ENTY=@LASTREC('DSTRLIST')('DSTRLIST'))

(706)   ADV($'STDIST')   ADJ('DISTRIBU')   -->
                 RECORD(SUP(ADV),LASTREC('DSTRLIST')=LASTREC('DSTRLIST')+1,
                        @LASTREC('DSTRLIST')('DSTRLIST')=RECORD),
                 ADJPH(SUP(ADV),ENTY=@LASTREC('DSTRLIST')('DSTRLIST'))

(707)   ADJPH($'STDIST',ENTY)   OPTCOMMA   PREP('WITH')   -->   ADJPHW(%ADJPH)

(708)   ADJPHW   NOUNPH($'ATTR',OBJ$'VALU')   -->
                 ADJPHW1(%ADJPHW,@SUP(NOUNPH)(ENTY)=ENTY(OBJ(NOUNPH))),
                 ADJPH(%ADJPHW)

(709)   ADJPHW1   OPTCOMMA   AND   -->   ADJPHW(%ADJPHW1)

(710)   PREP('FROM')   NOUNPH('UNIFORM')   -->   NOUNPH
(711)   PREP   NOUNPH   -->   PREPPH(%PREP,OBJ=NOUNPH)

(712)   PREPPH($'LOCDESCR',ENTY(OBJ)$'STATENTY')   -->
                 ADVIAL(%PREPPH,LOCOBJ=ENTY(OBJ),-OBJ,LASTLD(MEM)=SUP,
                        ATTRIB='LOCATION')
(713)   PREPPH('FOR',OBJ$'ABSTIME')   -->
                 ADVIAL(%OBJ(PREPPH),ATTRIB='DURATION')
(714)   PREPPH('IN',OBJ$'ABSTIME')   -->
                 ADVIAL(%OBJ(PREPPH),ATTRIB='DURATION')
(715)   PREPPH('EVERY',OBJ$'ABSTIME')   -->
                 ADVIAL(%OBJ(PREPPH),ATTRIB='IETM')
(716)   PREPPH('BY',ENTY(OBJ)$'ENTITY',ENTY(OBJ)¬$'STATENTY')   -->
                 ADVIAL(%OBJ(PREPPH),ATTRIB='AGENT')

(717)   CONJ('BUT',SUP='AND',1.NE.1)   -->   NULL
(718)   CONJ('UNTIL')   ATRCL(ATTRIB.EQ.'STATE')   -->
                 ADVIAL('COND1',CONDENTY=ENTY(ATRCL),ATTRIB='DURATION',
                        ENTY(VAL(ATRCL)).EQ.'UNAVAIL','COND2')

(719)   ADV('THERE',LASTSE(MEM))   -->
                 ADVIAL(SUP=LASTLD(MEM),LOCOBJ=LASTSE(MEM),ATTRIB='LOCATION',
                        ¬SUP,'AT')
(720)   ADV('THEN')   -->   ADV('FILLER',¬PRED(MEM),PRED(MEM)=LASTACT(MEM))

(721)   ADVIAL(¬ENTY)   -->   SRCHREC*1(%ADVIAL,LIST='MISCLIST',
                                        LC=10,TYPE='ADVIAL')
(722)   ADVIAL(ENTY,QUESTSW(MEM),ATTRIB.EQ.ATTRIB(MEM)|
             ATTRIB(MEM).EQ.'SUCARG'|ATTRIB(MEM).EQ.'PROBTIME')   -->
                 REPLYPH(%ADVIAL)


(723)   SRCHREC(LC.EQ.LASTREC(LIST))   -->
                 RECORD(%SRCHREC,LASTREC(LIST)=LC+1,
                        @LASTREC(LIST)(LIST)=RECORD,-LIST,-LC,-ENTY,-TYPE),
                 SRCHREC1*1(%SRCHREC,ENTY=@LASTREC(LIST)(LIST))
(724)   SRCHREC(LC=LC+1,ENTY=@LC(SEG)(LIST),SUP.NE.SUP(ENTY))   -->   SRCHREC
(725)   SRCHREC(NUM.NE.NUM(ENTY))   -->   SRCHREC
(726)   SRCHREC(LOCOBJ.NE.LOCOBJ(ENTY))   -->   SRCHREC
(727)   SRCHREC(OBJREL.NE.OBJREL(ENTY))   -->   SRCHREC
(728)   SRCHREC(CONDENTY.NE.CONDENTY(ENTY))   -->   SRCHREC
(729)   SRCHREC   -->   SRCHREC1(%SRCHREC)

(730)   SRCHREC1(TYPE.EQ.'NOUNPH')   -->
                 NOUNPH(%SRCHREC1,MENTY.NE.0,DECENTY=ENTY,ENTY=MENTY,
                        -MENTY,-MENTY(DECENTY),-DECENTY(DECENTY))
(731)   SRCHREC1(TYPE.EQ.'ADJPH')   -->   ADJPH(%SRCHREC1)
(732)   SRCHREC1(TYPE.EQ.'ADVIAL')   -->   ADVIAL(%SRCHREC1,-ATTRIB(ENTY))
(733)   SRCHREC1(TYPE.EQ.'NNPHUU')   -->   NNPHUU*1(%SRCHREC1)
(734)   SRCHREC1(TYPE.EQ.'ATRCL2')   -->   ATRCL2(%SRCHREC1,-LIST,-LC,-TYPE)


(735)   VERB('ENTER'|'LEAV')/   NOUNPH(ENTY$'STATENTY')   -->
                 ADVIAL('AT',LOCOBJ=ENTY(NOUNPH),LASTLD(MEM)='AT',
                        ATTRIB='LOCATION')
(736)   VERB('BECOM')   -->   VERB('BE')
(737)   VERB(DO,PLUR)   -->   VERBPH('DO',INF)
```

```
(738)     VERB('TAK')    -->    VERB('TAKTIME')
(739)     VERB('TAK')    NOUN('PLACE',SING)    -->    VERBPH(%VERB,'OCCUR')

(740)     VERB($'ACTION')    -->    VERB1*1(%VERB,LR=10,LC=LASTREC('ACTNLIST'))
(741)     VERB(¬$'ACTION',¬MODAL)    -->    VERBPH(%VERB)

(742)     VERB1(LC.EQ.10)    -->    VERBPH(%VERB1,LR.EQ.11,ENTY=@11)
(743)     VERB1(SUP.EQ.SUP(@LC(SEG)('ACTNLIST')))    -->
                VERB1(LR=LR+1,@LR=@LC(SEG)('ACTNLIST'),LC=LC-1)
(744)     VERB1    -->    VERB1(LC=LC-1)

(745)     VERBPH(¬PRM,'BE',¬SUBJECT,¬PREDICATE)    NOUNPH(¬NOUNAL,¬LOCATION) -->
                VERBPH(SUBJECT=NOUNPH,INTERG,THERE)
(746)     VERBPH(¬PRM,'BE',¬PREDICATE)    ADJPH    -->    VERBPH(PREDICATE=ADJPH)
(747)     VERBPH(¬PRM,'BE',¬PREDICATE)    NOUNPH(¬NOUNAL)    -->
                VERBPH(PREDICATE=NOUNPH)
(748)     VERBPH(¬PRM,'BE',¬PREDICATE)    ADVIAL(ENTY,ATTRIB.EQ.'LOCATION')    -->
                VERBPH(PREDICATE=ADVIAL)
(749)     VERBPH(¬PRM,'BE',¬PREDICATE)    ADV('NOT')    -->
                VERBPH(NEG=NEG+1)
(750)     VERBPH(¬PRM,'BE',¬SUBJECT,¬PREDICATE)    ADV('THERE')    -->
                VERBPH(INTERG)
(751)     VERBPH(¬PRM,'BE',¬SUBJECT,PREDICATE)    NOUNPH($'ATTR',OBJ)    -->
                VERBPH(SUBJECT=NOUNPH,INTERG)
(752)     ADJPH    VERBPH(¬PRM,'BE',SUBJECT$'ATTR',¬PREDICATE)    -->
                VERBPH(PRM,PREDICATE=ADJPH,ADJPH¬$'INTRGVAL',-INTERG)

(753)     VERBPH(¬PRM,'TAKTIME',OBJECT,¬SUBJECT)    VERBPH(TOINF)    -->
                VERBPH(%VERBPH#1,SUBJECT=VERBPH#2)
(754)     NOUNPH    VERBPH('TAKTIME',SUBJECT)    -->
                VERBPH(PRM,-INTERG,-THERE)
(755)     NOUNPH('IT'|'THAT')    VERBPH('TAKTIME',¬SUBJECT)    -->
                VERBPH(PRM,SUBJECT=%NOUNPH,-INTERG,-THERE,
                       -ENTY(SUBJECT),LASTREC('ACTNLIST').GT.10,
                       ENTY(SUBJECT)=@LASTREC('ACTNLIST')('ACTNLIST'))
(756)     VERBPH(TOINF)    VERBPH('TAKTIME',¬SUBJECT)    -->
                VERBPH(%VERBPH#2,PRM,SUBJECT=VERBPH#1,-INTERG,-THERE)

(757)     VERBPH(¬PRM,$'ACTION')    ADVIAL(ENTY,¬@ATTRIB(SEG)(VERBPH))    -->
                VERBPH1(%VERBPH,NEWATTR=ATTRIB(ADVIAL),
                       @NEWATTR=ENTY(ADVIAL))
(758)     VERBPH(¬PRM,TRANS*,¬OBJECT,¬GOAL)    NOUNPH    -->
                VBPHOBJ*1(%VERBPH,OBJECT=NOUNPH)

(759)     VERBPH(¬PRM,PRESPART,$'ACTION')    -->    VERBPH(-VFORM,NOUNAL)
(760)     VERBPH(NOUNAL,¬AGENT)    PREPPH('OF')    -->
                VERBPH1(%VERBPH,AGENT=ENTY(OBJ(PREPPH)),NEWATTR='AGENT')
(761)     VERBPH(NOUNAL,¬GOAL)    PREPPH('OF')    -->
                VERBPH1(%VERBPH,GOAL=ENTY(OBJ(PREPPH)),NEWATTR='GOAL')
(762)     NOUNPH    VERBPH(NOUNAL,¬AGENT,POSS(NOUNPH)|S)    -->
                VERBPH1(%VERBPH,PRM,AGENT=ENTY(NOUNPH),NEWATTR='AGENT')
(763)     NOUNPH    VERBPH(NOUNAL,¬GOAL,POSS(NOUNPH)|S)    -->
                VERBPH(%VERBPH,PRM,GOAL=ENTY(NOUNPH),NEWATTR='GOAL')
(764)     VERBPH(NOUNAL)    -->    NOUNPH(%VERBPH)
(765)     VERBPH(ENTY,INF|NOUNAL)    NOUN('TIME')    -->
                NOUNPH('TIME',PRM,OBJ=VERBPH)

(766)     VERBPH(¬PRM,¬NOUNAL)    ADV($'FILLER')    -->    VERBPH
(767)     VERBPH(¬PRM,INFF)    -->    VERBPH(PRM,-FINITE,INF)
(768)     VERBPH(¬PRM,PASTF)    -->    VERBPH(PRM,-PASTPART,PAST)

(769)     VERB('BE')    VERBPH(PASTPART,¬PASSIVE,¬PERFECT,¬PROG)    -->
                VERBPH(PRM,PASSIVE,VFORM=VFORM(VERB),INTERG,THERE,
                       ¬FINITE,-INTERG,-THERE)
(770)     VERB('BE')    VERBPH(PRESPART,¬PERFECT,¬PROG)    -->
                VERBPH(PRM,PROG,VFORM=VFORM(VERB),INTERG,THERE,
                       ¬FINITE,-INTERG,-THERE)
(771)     VERB('HAV')    VERBPH(PASTPART,¬PERFECT)    -->
                VERBPH(PRM,PERFECT,VFORM=VFORM(VERB),INTERG,
                       ¬FINITE,-INTERG)
(772)     VERB(MODAL)    VERBPH(INF)    -->
                VERBPH(PRM,MODAL=MODAL(VERB),VFORM=VFORM(VERB),INTERG)

(773)     ADV('NOT')    VERBPH(¬FINITE)    -->    VERBPH(PRM,NEG=NEG+1)
(774)     ADV('THERE')    VERBPH(THERE)    -->    VERBPH(PRM,-INTERG,-THERE)
(775)     ADV('WHAT')    VERBPH('DO',SUBJECT)    -->    VERBPH(PRM)
(776)     ADV($'INTRGVAL')    VERBPH(SUBJECT,INTERG,$'ACTION')    -->
                VERBPH(PRM,ATTRIB($(ADV)))
(777)     ADV($'FILLER')    VERBPH(¬NOUNAL)    -->    VERBPH(PRM)

(778)     NOUNPH    VERBPH(¬TOINF,¬NOUNAL,¬SUBJECT,¬AGENT|¬GOAL)    -->
                VBPHSUBJ*1(%VERBPH,PRM,SUBJECT=NOUNPH,-INTERG,-THERE)
(779)     ADVIAL(ENTY,ATTRIB.EQ.'LOCATION')
                          VERBPH(¬'BE',¬LOCATION,SUBJECT,¬NOUNAL)    -->
                VERBPH1(%VERBPH,PRM,LOCATION=ENTY(ADVIAL),
                       NEWATTR='LOCATION')
(780)     ADVIAL(ENTY,ATTRIB.NE.'LOCATION')
                          VERBPH(¬@ATTRIB(ADVIAL),¬TOINF,¬NOUNAL)    -->
                VERBPH1(%VERBPH,PRM,NEWATTR=ATTRIB(ADVIAL),
                       @NEWATTR=ENTY(ADVIAL))
(781)     PREP('TO')    VERBPH(INF,¬SUBJECT)    -->
                VERBPH(PRM,TOINF,-INF,-INTERG,-THERE)
(782)     PREPPH('FOR')    VERBPH(TOINF,¬SUBJECT)    -->
                VBPHSUBJ(%VERBPH,SUBJECT=OBJ(PREPPH))
```

```
(783)   VBPHOBJ(¬$'ACTION')   -->   VERBPH(%VBPHOBJ)
(784)   VBPHOBJ(ENTY(OBJECT)$'ENTITY',ENTY(OBJECT)¬$'STATENTY',¬GOAL)   -->
                 VERBPH1(%VBPHOBJ,GOAL=ENTY(OBJECT),-OBJECT,
                         NEWATTR='GOAL')

(785)   VBPHSUBJ(¬$'ACTION')   -->   VERBPH(%VBPHSUBJ)
(786)   VBPHSUBJ(ENTY(SUBJECT)$'ENTITY',ENTY(SUBJECT)¬$'STATENTY',
                 ¬AGENT,¬PASSIVE)   -->
                 VERBPH1(%VBPHSUBJ,AGENT=ENTY(SUBJECT),NEWATTR='AGENT')
(787)   VBPHSUBJ(ENTY(SUBJECT)$'ENTITY',ENTY(SUBJECT)¬$'STATENTY',¬GOAL,
                 PASSIVE|PASTPART)   -->
                 VERBPH1(%VBPHSUBJ,GOAL=ENTY(SUBJECT),NEWATTR='GOAL')

(788)   VERBPH1   -->   VERBPH2*1(%VERBPH1,LC=11,LRC=10,ENTY=@11)

(789)   VERBPH2(LC.GT.LR)   -->   VERBPH3*1(%VERBPH2,LC=LRC+1)
(790)   VERBPH2(¬@NEWATTR(SEG)(ENTY)|@NEWATTR.EQ.@NEWATTR(SEG)(ENTY))   -->
                 VERBPH2(LRC=LRC+1,@LRC=@LC,LC=LC+1,ENTY=@LC)
(791)   VERBPH2   -->   VERBPH2(LC=LC+1,ENTY=@LC)

(792)   VERBPH3(LC.GT.LR)   -->   VERBPH(%VERBPH3,LR=LRC,-LC,-LRC,-ENTY,
                    -NEWATTR,LR.EQ.11,ENTY=@11)
(793)   VERBPH3   -->   VERBPH3(-@LC,LC=LC+1)


(794)   #   -->   PUNCB
(795)   ,   -->   DELIMA,   DELIMB,   DELIMC,   PUNCB
(796)   .   -->   DELIMB,   DELIMC,   PUNCA,   PUNCB
(797)   ;   -->   DELIMB,   DELIMC,   PUNCA,   PUNCB
(798)   ?   -->   DELIMB,   DELIMC,   PUNCA,   PUNCB

(799)   CONJ('AND')   -->   DELIMA,   DELIMB,   DELIMC
(800)   CONJ('IF'|ATTRIB(SUP))   -->   DELIMB,   DELIMC
(801)   ADV('OTHERWIS')   -->   DELIMB,   DELIMC
(802)   CONJ('UNTIL')   -->   DELIMC

(803)   CONJ('AND')   /#   -->   AND
(804)   .   /#   -->   PERIOD(PERIODSW(MEM)=1)
(805)   ;   /#   -->   PERIOD(-PERIODSW(MEM))
(806)   ?   /#   -->   QUESMARK
(807)   ,   /#   -->   COMMA,   OPTCOMMA,   KWORD
(808)   /#   -->   OPTCOMMA

(809)   PERIOD   -->   KWORD('ENDOFSENT')
(810)   QUESMARK   -->   KWORD('ENDOFSENT')

(811)   DELIMC/   VERBPH($'ATTRVERB',SUBJECT,FINITE)   /DELIMB   -->
                 ATRCL1*1(%VERBPH)

(812)   DELIMB/   VERBPH(INTERG,FINITE,$'ACTION'|'DO')   /DELIMB   -->
                 ACTCL(%VERBPH,ENTY=@11)

(813)   DELIMB/   VERBPH($'ACTION',¬INTERG,FINITE|PRESPART)   /DELIMB   -->
                 ACTCL1*1(%VERBPH)

(814)   DELIMB/   VERBPH(DURATION$'COND',¬INTERG,FINITE|PRESPART)   -->
                 ACTCL1(%VERBPH)

(815)   DELIMB/   VERBPH(NOUNAL(SUBJECT),'OCCUR'|'HAPPEN')   -->
                 VERBPH(%SUBJECT,VERBPHIND=VERBPHIND(VERBPH),-PRM)

(816)   ATRCL1(SUBJECT$'ATTR',ENTY(PREDICATE)$'VALU')   -->
                 ATRCL(%OBJ(SUBJECT(ATRCL1)),ATTRIB=SUP(SUBJECT(ATRCL1)),
                       VAL=PREDICATE(ATRCL1),NEG=NEG(ATRCL1))
(817)   ATRCL1(ENTY(SUBJECT)$'ENTITY',ATTRIB($(PREDICATE)))   -->
                 ATRCL(%SUBJECT(ATRCL1),VAL=PREDICATE(ATRCL1),ATTRIB($(VAL)),
                       NEG=NEG(ATRCL1))
(818)   ATRCL1(DECENTY(SUBJECT),ENTY(PREDICATE)$'ENTITY')   -->
                 ATRCL(DECENTY(SUBJECT(ATRCL1)),ENTY(PREDICATE(ATRCL1)),
                       STRUCENTY=ENTY(SUBJECT(ATRCL1)))
(819)   ATRCL1(ENTY(SUBJECT)$'ACTION',PREDICATE$'STDIST')   -->
                 ATRCL(%SUBJECT(ATRCL1),ATTRIB='TIME',VAL=PREDICATE(ATRCL1),
                       NEG=NEG(ATRCL1))
(820)   ATRCL1('TAKTIME',SUBJECT$'ACTIVITY',OBJECT$'VALU')   -->
                 ATRCL(%SUBJECT(ATRCL1),ATTRIB='DURATION',VAL=OBJECT(ATRCL1),
                       NEG=NEG(ATRCL1))
(821)   ATRCL1('HOLD')   -->   RECORD('CAPACITY',OBJ=SUBJECT(ATRCL1),
                       RP(MEM)=RECORD),
                 ATRCL1('BE',SUBJECT=RP(MEM),PREDICATE=OBJECT,
                       -OBJECT,-RP(MEM))
(822)   ATRCL1(SUP(SUBJECT).EQ.'CAPACITY',PREDICATE$'ENTITY')   -->
                 ATRCL(%OBJ(SUBJECT(ATRCL1)),ATTRIB='CAPACITY',
                       VAL=QUANTITY(PREDICATE(ATRCL1)),NEG=NEG(ATRCL1))
(823)   ATRCL1('BE',ENTY(SUBJECT)$'ENTITY',QUANTITY(SUBJECT),¬PREDICATE)   -->
                 ATRCL(%SUBJECT(ATRCL1),ATTRIB='QUANTITY',VAL=QUANTITY,
                       NEG=NEG(ATRCL1))
(824)   ATRCL1('HAV',ENTY(SUBJECT)$'STATENTY',ENTY(OBJECT)$'STATENTY')   -->
                 SRCHREC('IN',LOCOBJ=ENTY(SUBJECT(ATRCL1)),SAVEREC=ATRCL1,
                         LIST='MISCLIST',LC=10,TYPE='ATRCL2')
```

```
(B25)     ATRCL2   -->    ATRCL(%OBJECT(SAVEREC(ATRCL2)),ATTRIB='LOCATION',
                          VAL=ATRCL2,NEG=NEG(SAVEREC(VAL)),-SAVEREC(ENTY(VAL)))

(B26)     ACTCL1(LR.EQ.10)    -->
                 RECORD(%ACTCL1,-LR,-LC,-LRC,-ENTY,-SUBJECT,-OBJECT,
                        LASTREC('ACTNLIST')=LASTREC('ACTNLIST')+1,
                        IDNO=LASTREC('ACTNLIST')-10,-CHECKED('ACTNLIST'),
                        @LASTREC('ACTNLIST')('ACTNLIST')=RECORD),
                 ACTCL(%ACTCL1,ENTY=@LASTREC('ACTNLIST')('ACTNLIST'))
(827)     ACTCL1   -->    ACTCL2*1(%ACTCL1,ENTY=@11)

(B28)     ACTCL2(AGENT,¬AGENT(ENTY))    -->    ACTCL2(AGENT(ENTY)=AGENT,
                                                    -CHECKED(ENTY))
(829)     ACTCL2(GOAL,¬GOAL(ENTY))    -->   ACTCL2(GOAL(ENTY)=GOAL,-CHECKED(ENTY))
(B30)     ACTCL2(LOCATION,¬LOCATION(ENTY))   --> ACTCL2(LOCATION(ENTY)=LOCATION,
                                                    -CHECKED(ENTY))
(B31)     ACTCL2(DURATION,¬DURATION(ENTY))   --> ACTCL2(DURATION(ENTY)=DURATION,
                                                    -CHECKED(ENTY))
(B32)     ACTCL2(IETM,¬IETM(ENTY))    -->   ACTCL2(IETM(ENTY)=IETM,-CHECKED(ENTY))
(833)     ACTCL2   -->    ACTCL(%ACTCL2)

(834)     CONJ(ATTRIB(SUP))  ACTCL    -->    SUBCL(%ACTCL,ATTRIB(SUP(CONJ)))
(835)     CONJ('IF')    ATRCL(ATTRIB.EQ.'LENGTH'|ATTRIB.EQ.'STATE')    -->
                 CONDCL1*1(%ATRCL)

(B36)     CONDCL1(ATTRIB.EQ.'LENGTH',SUP(VAL).EQ.'LT')    -->
                 CONDCL2*1('QTYP',SUCARG=LOCOBJ(LOCATION(ENTY(CONDCL1))),
                        MAXQ=OBJREL(VAL(CONDCL1)),ACT1='OPENACT',
                        ACT2='CLOSACT',ATTRIB='CONDITN',NEG=NEG(CONDCL1))
(837)     CONDCL1(ATTRIB.EQ.'STATE',ENTY(VAL).EQ.'AVAILABL'|
                 ENTY(VAL).EQ.'FREE')    -->    CONDCL1(-VAL,NEG=NEG+1)
(83B)     CONDCL1(ATTRIB.EQ.'STATE')    -->
                 CONDCL2('FTYP',SUCARG=ENTY(CONDCL1),ACT1='CLOSACT',
                        ACT2='OPENACT',ATTRIB='CONDITN',NEG=NEG(CONDCL1))

(B39)     CONDCL2(NEG)    -->
                 SUBCL(%CONDCL2,ACT3=ACT1,ACT1=ACT2,ACT2=ACT3,-ACT3,-NEG)
(B40)     CONDCL2   -->    SUBCL(%CONDCL2)

(B41)     ADV($'FILLER')    -->    SUBCL(%ADV,ATTRIB='AFILLER')
(842)     ADV('OTHERWIS')    -->    SUBCL(%ADV,ATTRIB='CONDITN')


(843)     PUNCA/    -->    SENT(LR=10)
(844)     SENT(¬SUP|'ACTION')    ACTCL    -->
                 SENT('ACTION',LR=LR+1,@LR=ACTCL,LASTACT(MEM)=ENTY(ACTCL))
(845)     SENT(¬SUP|'ACTION')    SUBCL(¬@ATTRIB(SEG)(SENT))    -->
                 SENT('ACTION',@ATTRIB(SUBCL)=SUBCL)
(B46)     SENT(¬SUP|'ATTRVERB')    ATRCL    -->
                 SENT('ATTRVERB',LR=LR+1,@LR=ATRCL)
(B47)     SENT(SUP)    COMMA    -->    SENT
(84B)     SENT(LR.GT.10)    AND    -->    SENT

(B49)     PUNCA/         -->    KWSENT
(B50)     KWSENT(¬ENDOFSENT)    KWORD    -->
                 KWSENT(SPARSED=SPARSED(MEM),-SPARSED(MEM),LR=LR+1,
                        SUP(KWORD).NE.0,@SUP(KWORD)=LR,
                        SUP(KWORD).EQ.'TIMENTY',TIMENTY=ENTY(KWORD))


SEMOLOGY FOR DECODING ENGLISH:

(B51)     PUNCA/    REPLYPH    PERIOD    -->    REPLY*1(%REPLYPH,SPARSED(MEM))

(B52)     REPLY(ATTRIB(MEM),CENTY(MEM),ENTY,¬SUPSET(MEM)|ENTY$SUPSET(MEM)) -->
                 OK(@ATTRIB(MEM)(CENTY(MEM))=ENTY(REPLY),
                     ATTRIB(MEM).EQ.'SUCARG',
                     SUCARG(CENTY(MEM))=LOCOBJ(ENTY(REPLY)))
(853)     REPLY    -->    ENCODING(ETYPE='REPLYERR')


(B54)     SENT('ACTION',LR.GT.10)    PERIOD    -->
                 ACTSENT*1(%SENT,LC=11,XYLAST=100,SPARSED(MEM))
(B55)     SENT('ATTRVERB')    PERIOD    -->
                 ATRSENT*1(%SENT,LC=11,XYLAST=100,SPARSED(MEM))
(B56)     SENT    QUESMARK    -->
                 QUESTION*1(%SENT,LC=11,XYLAST=100,SPARSED(MEM))


(857)     ATRSENT(DECENTY(@11))    -->
                 ATRSENT2(%ATRSENT,STRUCENTY(@11),CLASATR(STRUCENTY)='SOUP')
(B5B)     ATRSENT(LC.LE.LR)    -->    ATRSENT1*1(%@LC(ATRSENT)(ATRSENT)),
                                      ATRSENT(LC=LC+1)

(B59)     ATRSENT1(¬'DECIMAL',ENTY,ENTY(VAL),¬NEG)    -->
                 OK(%ATRSENT1,@ATTRIB(SEG)(ENTY)=ENTY(VAL),-CHECKED(ENTY),
                     QUANTITY.NE.0,QUANTITY(ENTY)=ENTY(QUANTITY))
```

```
(860)   ATRSENT1   -->   ERROR(MESSAGE3)

(861)   ATRSENT2(LC,LC.LE.LR)   -->
                ATRSENT2(STRUC(ENTY(@LC))=STRUCENTY,XYLAST=XYLAST+1,
                     @XYLAST=%'REST',-NAME(@XYLAST),
                     CPC=CPC+NUM(DECENTY(@LC)),NUM(@XYLAST)=CPC,
                     XYLAST=XYLAST+1,@XYLAST=ENTY(@LC),-@LC,LC=LC+1,
                     LC.GT.LR,-CPC,-LR,-LC,'TYPDIST',PNUM='ONE',
                     FNARG='RANDMREC',DORC="D",ASNDISTR(MEM)=RECORD,
                     LASTREC('DSTRLIST')=LASTREC('DSTRLIST')+1,
                     @LASTREC('DSTRLIST')('DSTRLIST')=RECORD)

(862)   ACTSENT(SUCC)   -->   ACTSENT(SUCC(MEM)=ENTY(SUCC),-SUCC)
(863)   ACTSENT(PRED)   -->   ACTSENT(PRED(MEM)=ENTY(PRED),-PRED)
(864)   ACTSENT(CONDITN)   -->   ACTSENT(CONDITN(MEM)=CONDITN,-CONDITN)
(865)   ACTSENT(LR.EQ.11)   -->   ACTSENT4*1(SUCCVAL=ENTY(@11(ACTSENT)))
(866)   ACTSENT(DECENTY(SUBJECT(@11)))   -->   ACTSENT1*1(%ACTSENT)
(867)   ACTSENT(ENTY(SUBJECT(@11)).NE.ENTY(SUBJECT(@12)))   -->
                ACTSENT2*1(%ACTSENT)
(868)   ACTSENT   -->   ACTSENT3*1(%ACTSENT)

(869)   ACTSENT1(LC.LE.LR,¬DECENTY(SUBJECT(@LC))|
              ENTY(SUBJECT(@LC)).NE.ENTY(SUBJECT(@LR)))   -->   ERROR3
(870)   ACTSENT1(LC.LE.LR)   -->
                ACTSENT1(XYLAST=XYLAST+1,@XYLAST=%'REST',-NAME(@XYLAST),
                     CPC=CPC+NUM(DECENTY(SUBJECT(@LC))),
                     NUM(@XYLAST)=CPC,XYLAST=XYLAST+1,@XYLAST=ENTY(@LC)
                     -@LC,LC=LC+1)
(871)   ACTSENT1   -->
                RECORD(%ACTSENT1,-LC,-LR,
                     -CPC,-ATHEN,'FRACTNL',SUCARG='RANDMREC',
                     LASTREC('SCSRLIST')=LASTREC('SCSRLIST')+1,
                     @LASTREC('SCSRLIST')('SCSRLIST')=RECORD),
                ACTSENT4(SUCCVAL=@LASTREC('SCSRLIST')('SCSRLIST'))

(872)   ACTSENT2(LC.NE.LR,ENTY(SUBJECT(@LC)).EQ.ENTY(SUBJECT(@LR)))   -->
                ERROR3
(873)   ACTSENT2(LC.LT.LR)   -->
                ACTSENT2(XYLAST=XYLAST+1,@XYLAST=ENTY(SUBJECT(@LC)),
                     XYLAST=XYLAST+1,@XYLAST=ENTY(@LC),-@LC,LC=LC+1)
(874)   ACTSENT2   -->
                RECORD(%ACTSENT2,XYLAST=XYLAST+1,@XYLAST=ENTY(SUBJECT(@LC)),
                     XYLAST=XYLAST+1,@XYLAST=ENTY(@LC),-@LC,-LC,-LR,
                     -ATHEN,'PTYP',SUCARG='PARAM1',
                     LASTREC('SCSRLIST')=LASTREC('SCSRLIST')+1,
                     @LASTREC('SCSRLIST')('SCSRLIST')=RECORD),
                ACTSENT4(SUCCVAL=@LASTREC('SCSRLIST')('SCSRLIST'))

(875)   ACTSENT3(ENTY(SUBJECT(@LC)).NE.ENTY(SUBJECT(@LR)))   -->   ERROR3
(876)   ACTSENT3(LC.LT.LR)   -->
                SETSUCC(LC1=LC(ACTSENT3)+1,ENTY(@LC(ACTSENT3)(ACTSENT3)),
                     SUCCVAL=ENTY(@LC1(SEG)(ACTSENT3))),
                ACTSENT3(LC=LC+1)
(877)   ACTSENT3   -->   ACTSENT4(SUCCVAL=ENTY(@11(ACTSENT3)))

(878)   ACTSENT4(QUESTSW(MEM),ATTRIB(MEM).EQ.'OPENACT'|
                ATTRIB(MEM).EQ.'CLOSACT')   -->
                OK(@ATTRIB(MEM)(CENTY(MEM))=SUCCVAL(ACTSENT4),
                   -ACT2(CENTY(MEM)),-CONDITN(MEM),-PRED(MEM))
(879)   ACTSENT4(SUP(CONDITN(MEM)).EQ.'OTHERWIS')   -->
                ACTSENT5*1(%ACTSENT4,LC=LC(MEM),¬LC(MEM),
                     LC=LASTREC('SCSRLIST'))
(880)   ACTSENT4(CONDITN(MEM))   -->
                ACTSENT4(CONDITN(MEM),@ACT1(CONDITN(SEG))(CONDITN)=SUCCVAL,
                     -ACT1(CONDITN),-ATTRIB(CONDITN),SUCCVAL=CONDITN,
                     LASTREC('SCSRLIST')=LASTREC('SCSRLIST')+1,
                     @LASTREC('SCSRLIST')('SCSRLIST')=CONDITN,
                     -CONDITN(MEM))
(881)   ACTSENT4(PRED(MEM))   -->
                SETSUCC*1(ENTY=PRED(MEM),SUCCVAL(ACTSENT4))
(882)   ACTSENT4(SUCC(MEM))   -->
                SETSUCC(ENTY=SUCCVAL(ACTSENT4),SUCCVAL=SUCC(MEM))
(883)   ACTSENT4   -->   OK

(884)   ACTSENT5(LC.LT.11)   -->   ERROR3
(885)   ACTSENT5(ACT2(@LC(SEG)('SCSRLIST')))   -->
                OK(CONDITN=@LC(ACTSENT5)('SCSRLIST'),
                   @ACT2(CONDITN(SEG))(CONDITN)=SUCCVAL(ACTSENT5),
                   -ACT2(CONDITN),-CONDITN(MEM),-PRED(MEM),-LC(MEM))
(886)   ACTSENT5   -->   ACTSENT5(LC=LC-1)

(887)   SETSUCC(SUCC(ENTY))   -->   SETSUCC(-SUCC(ENTY),SUCCCHANGED(MEM))
(888)   SETSUCC(LC1)   -->
                DUMMY(%SETSUCC,SUCC(ENTY)=SUCCVAL,-CHECKED(ENTY))
(889)   SETSUCC   -->   OK(%SETSUCC,SUCC(ENTY)=SUCCVAL,-CHECKED(ENTY),
                     -PRED(MEM),-SUCC(MEM))

(890)   KWSENT(ENDOFSENT,¬SPARSED)   -->   KWDSENT*1(%KWSENT)

(891)   KWDSENT(TIMENTY,PROBLEM|RUN|SIMULATE)   -->
                OK(PROBTIME(MEM)=TIMENTY(KWDSENT))
(892)   KWDSENT(TIMENTY,TIME,UNIT)   -->   OK(TIMUNIT(MEM)=TIMENTY(KWDSENT))
(893)   KWDSENT(ENGLISH|STATE|DESCRIBE)   -->
```

```
                     ENCODING(ETYPE(MEM)='ENGLISH')
(894)      KWDSENT(GPSS|PROGRAM)    -->    ENCODING(ETYPE(MEM)='GPSSPROG')
(895)      KWDSENT(ASK|QUESTION|PROMPT|INQUIRY)    -->
                     ENCODING(ETYPE(MEM)='QUESANSW')
(896)      KWDSENT(OK|OKAY|COMPLETE|DONE|CHECK)    -->
                     ENCODING(ETYPE(MEM)='CHECKIPR')
(897)      KWDSENT(EXPLAIN|CLARIFY|WHAT|WHICH)    -->
                     ENCODING(ETYPE(MEM)='CHECKIPR')
(898)      KWDSENT(STOP)    -->    OK(-QUESTSW(MEM),-PRED(MEM),-SUCC(MEM),
                                   -CENTY(MEM),-ATTRIB(MEM),¬SUPSET(MEM),
                                   -CONDITN(MEM),¬QAMODE(MEM))
(899)      KWDSENT    -->    ERROR(MESSAGE2)

(900)      ERROR3    -->    ERROR(MESSAGE3,¬QUESTSW(MEM),-CONDITN(MEM),-PRED(MEM),
                            ¬SUCC(MEM))

(901)      QUESTION    -->    ENCODING(%QUESTION,ETYPE='QUESTION'),    OK


(902)      OK(QUESTSW(MEM),PERIODSW(MEM))    -->    ENCODING
(903)      ERROR    -->    NULL
(904)      RECORD    -->    NULL
(905)      DUMMY    -->    NULL
(906)      ENCODING    -->    NULL

:END OF FILE:
```

APPENDIX J

Alphabetical listings of the names of

named records,

indicators,

attributes, and

routines

that appear in the listings

in Appendices A-I

Named Record Names

A, ABSCLOCK, ABSCOLR, ABSQNTY, ABSTATE, ABSTIME, ABSWEIT,
ACPTR, ACTION, ACTIVITY, ACTNLIST, ACT1, ACT2, ACT3, ADJ,
ADJPH, ADV, ADVANCE, ADVIAL, AENTY, AFILLER, AFTER, AGENT,
AGORGL, AN, AND, APTR, ARE, ARGAZ, ARGREC, ARGVAL, AROUND,
ARRIV, AS, ASET, ASK, ASNDISTR, ASSIGN, AT, ATHEN, ATNO,
ATNO2, ATRCL2, ATRPH2, ATTR, ATTRIB, ATTRIB1, ATTRVERB,
AVAILABL, BACKSTUF, BANK, BARGE, BE, BECAUSE, BECOM, BEFORE,
BETWEEN, BIG, BLACK, BLKDIR, BLUE, BTCODE, BUSY, BY, CAN,
CAPACITY, CAR, CARGO, CENTY, CHARS, CHECK, CHECKIPR, CLARIFY,
CLASATR, CLOSACT, CMPLX, COLOR, COMCNT, COMMENT, COMMERCE,
COMMISC, COMPLETE, COND, CONDENTY, CONDITN, COND1, COND2,
CONJ, CONSUMP, CPC, CUSTOMER, DARK, DATA, DAY, DECENTY,
DECIMAL, DECREC, DEPART, DEPOT, DESCRIBE, DET, DETERM, DEVIATIO,
DIRPTR, DISTPTR, DISTRIBU, DISTR1, DO, DOCK, DOES, DONE, DORC,
DSTRLIST, DURATION, EIGHT, EMPDIST, END, ENDOFSEN, ENGLISH,
ENTER, ENTITY, ENTY, EQ, EQU, EQUAL, ER, EST, ETYPE, EVENT,
EVERY, EXPLAIN, EXPON, EXPONREC, FAST, FEW, FILLER, FIVE, FMOD,
FNARG, FNCTN, FNDIR, FNTYPE, FOR, FOUR, FOURTH, FRACTNL, FREE,
FROM, FTYP, FULL, FUNCNO, FUNCTION, FVARIABL, GASSTA, GATE,
GENERATE, GO, GOAL, GOTO, GPSS, GPSSPROG, GRAY, GREATER, GREEN,
GRNNO, GT, HALF, HAPPEN, HARBOR, HAS, HAV, HE, HEAVY, HELD,
HER, HIM, HOLD, HOUR, HOW, HOWLONG, HOWOFTEN, IDK, IDNAME,
IDNO, IDSNO, IETM, IF, IMMEDIAT, IN, INQUIRY, INTRGPH, INTRGVAL,
INTWIDTH, IPDP, IS, IT, JUST, K, LABL, LARGE, LAST, LASTACT,
LASTE, LASTLD, LASTME, LASTREC, LASTSE, LAVENDER, LC, LC1, LEAV,
LEAVE, LEFT, LENGTH, LESS, LIGHT1, LIGHT2, LINE, LINEREC, LIST,
LITTLE, LOAD, LOCAT, LOCATION, LOCDESCR, LOCOBJ, LONG, LOWINT,
LR, LRC, LT, MAN, MANY, MARK, MAUVE, MAXQ, MBNTY, MEAN, MEN,
MENTY, MEPTR, MFNID, MINUTE, MISC, MISCLIST, MISCPTR, MOBENATR,
MOBENTY, MOBLIST, MOD, MVARID, NAM, NAM2, NCOM, NE, NEAR,
NEWATTR, NG, NINE, NL, NNPHUU, NO, NOPTS, NORMAL, NORMREC, NOT,
NOUN, NOUNPH, NOUNS, NULL, NUM, NUMINT, OBJ, OBJECT, OBJPPH,
OBJREL, OCCUR, OF, OFTEN, OK, OKAY, ON, ONE, OPENACT, ORANGE,
OTHER, OTHERWIS, OUNCE, OWNER, PARAMNO, PARAM1, PBMS, PC, PCPC,
PER, PERCENT, PERIODSW, PERMISC, PERS, PERSON, PERSONAL, PIER,
PLACE, PNUM, PORT, POUND, PRED, PREDADJ, PREDICAT, PREDNOM,
PREDVAL, PREP, PROBLEM, PROBTIME, PROGRAM, PROMPT, PRON, PS,
PTYP, PUMP, PURPLE, QDIR, QTYP, QUALVAL, QUANTITY, QUANVAL,
QUARTER, QUESANSW, QUESTION, QUEUE, RANDM, RANDMREC, RANGE, RATE,
REASON, REASON1, REASON2, REASON3, REC, RECLIST, REC2, RED,
RELCOLR, RELEASE, RELIND1, RELIND2, RELQNTY, RELSIZ, RELSPEED,
RELTIME, RELTV, RELVAL, RELWEIT, REPLYERR, REST, RIGHT, RMULT,
RNNO, RP, RP2, RUN, SAME, SAVEREC, SAVEVALU, SCSRLIST, SECOND,
SEEDS, SEIZE, SELECT, SENTT, SEPTR, SERVIC, SETMEM, SEVEN, SHE,
SHIP, SHOULD, SIMPLY, SIMULATE, SIMULATI, SIX, SIZE, SLOW, SMALL,
SMOD, SNAREC, SNAREF, SOMETHIN, SOUP, SPEED, STALIST, STANDARD,
START, STATE, STATENTY, STATION, STDEV, STDIST, STOP, STORAGE,
STORDIR, STORIND, STRUCENT, STYP, SUBJECT, SUCARG, SUCC, SUCCVAL,
SUCDESCR, SUCDSCR1, SUCDSCR2, SUCPTR, SUPSET, SVDIR, SYN, TABDIR,
TABL, TABLE, TABULATE, TAK, TAKTIME, TARGA, TEMP, TEN, TERMINAT,
TEST, THAN, THAT, THE, THEM, THEN, THERE, THESE, THEY, THIRD,
THIS, THREE, TIME, TIMENTY, TIMUNIT, TLAST, TO, TON, TOOK,
TRANSFER, TRANSREC, TRANSTIM, TRUCK, TWO, TYPDIST, TYPE, TYPEVAL,
TYPTABL, UNAVAIL, UNIFORM, UNIT, UNITLIST, UNITPTR, UNITS, UNLOAD,
UNTIL, VAL, VALQUAN, VALSET, VALU, VARDIR, VEHICLE, VERB, VERBS,
VIOLET, WAIT, WEIGHT, WHAT, WHATREC, WHEN, WHERE, WHICH, WHITE,
WILL, WINDOW, WITH, X, XMOD, XRNNO, XYC, XYLAST, Y, YELLOW, YES

Indicator Names (and Numbers)

AUXIL(22-27), CAN(24), CHECKED(49), DO(23), DOLLARSI(3), E(2), ED(6), EN(8), ER(7), ES(4), EXPNORM(13), EXPUSED(1), FINITE(13-15), FUTURE(22), GPSSSW(11), INCOMP(11), INF(16), INFF(20), INFIN(16), ING(5), INTERG(31), LIMITCHK(50), LY(1), MARK1(46), MARK2(47), MODAL(22-24), N(9), NEG(19), NOCOPY(14), NOREST(8), NORMUSED(2), NOUNAL(29), NSFX(1), NUMB(14-15), NUMBSUBJ(36-37), PASSIVE(25), PAST(13), PASTF(21), PASTPART(18), PERFECT(27), PERS1(11), PERS2(12), PLUR(15), POSS(13), PRESPART(17), PRM(33), PROG(26), PRONIND(11-15), PT(9-10), QAMODE(10), QUESMK(34), QUESTSW(7), R(10), REACHED(48), S(3), SING(14), SPARSED(5), SUCCCHAN(9), SUFX(1-10), SUNDERST(6), THERE(32), TOINF(28), TRANS(12), VERBPHIN(13-33), VERBPIND(13-24), VFORM(13-18), WRECOGNI(4), XVSW(12),

Attribute Names (and Numbers)

ARGA(11). ARGB(12), ARGC(13), ARGD(14), ARGE(15), ARGF(16), ARGG(17), ARGH(18), INDIC(2), IX(8), NAME(10), STRUC(9), SUP(1), XV(9)

Routine Names (and Numbers)

CONVERT(2), ENDOFWOR(3), FORMSYMB(4), GETYPE(10), LOOKUP(1), MESSAGE1(6),MESSAGE2(7), MESSAGE3(8), MESSAGE4(9),

LIST OF REFERENCES

1. Balzer, R. M., and Farber, D. J., "APAREL - a parse-request language," COMM. ACM 12, 11 (Nov. 1969), 624-631.

2. Chomsky, N., "Three models for the description of language," IEEE TRANS. INFO. THEORY, vol. IT-2, Proceedings of the Symposium on Information Theory, Sept. 1956.

3. Coles, L. S., "Syntax directed interpretation of natural language," Ph.D. Dissertation, Carnegie Mellon Univ., Pittsburgh, Penn., 1967.

4. Earley, J., "An efficient context-free parsing algorithm," COMM. ACM 13, 2 (Feb. 1970), 94-102.

5. Feldman, J. A., and Gries, D., "Translator writing systems," COMM. ACM 11, 2 (Feb. 1968), 77-113.

6. Floyd, R. W., "A descriptive language for symbol manipulation," J. ACM 8 (Oct. 1961), 579-584.

7. GENERAL PURPOSE SIMULATION SYSTEM /360 - INTRODUCTORY USER'S MANUAL, Publication H20-0304, IBM DP Div, White Plains, N.Y., 1967.

8. Irons, E. T., "A syntax directed compiler for ALGOL 60," COMM. ACM 4, 2 (Jan. 1961), 51-55.

9. Irons, E. T., "An error-correcting parse algorithm," COMM. ACM 6, 11 (Nov. 1963), 669-673.

10. Kaplan, R. M., "The MIND system: a grammar-rule language," MEMORANDUM RM-6265/1-PR, The RAND Corporation, Santa Monica, Calif., April 1970.

11. Klein, S., and Simmons, R. F., "Syntactic dependence and the computer generation of coherent discourse," MECH. TRANS. 7, 2 (Aug. 1963), 50-61.

12. Klein, S., "Automatic paraphrasing in essay format," MECH. TRANS. 8:3/4 (1965), 68-83.

13. Klein, S., "Control of style with a generative grammar," LANGUAGE 41, 4 (Oct.-Dec. 1965), 619-631.

14. Knuth, D. E., THE ART OF COMPUTER PROGRAMMING, Vol 1 Fundamental Algorithms, Addison-Wesley, Reading, Mass., 1968.

15. Knuth, D. E., "Semantics of Context-Free Languages," MATH. SYS. THEORY 2, 2 (June 1968), 127-145.

16. Knuth, D. E., "Examples of Formal Semantics" AI MEMO 126, Comp. Sci. Dept., Stanford Univ., Stanford, Calif., July 1970.

17. Lamb, S. M., "The sememic approach to structural semantics," AMERICAN ANTHROPOLOGIST 66:3 (pt 2) (June 1964), 57-78.

18. Lamb, S. M., OUTLINE OF STRATIFICATIONAL GRAMMAR, Revised edition, Georgetown Univ. Press, Washington, D. C., 1966.

19. Lamb, S. M., "Lexicology and semantics," In LINGUISTICS TODAY (A. Hill, ed.), Basic Books, New York, 1969, 40-49.

20. Lamb, S. M., "Linguistic and cognitive networks," In COGNITION: A MULTIPLE VIEW (P. Garvin, ed.), Spartan Books, New York, 1970, 195-222.

21. Lamb, S. M., "The crooked path of progress in cognitive linguistics," Monograph Series on Languages and Linguistics, No. 24, Report on the 22nd Annual Round Table Meeting, Georgetown Univ., 1971.

22. Lang, B., "Parallel non-deterministic bottom-up parsing," Center for Research in Computing Technology, Harvard Univ., Cambridge, Mass., Aug. 1971.

23. Lockwood, D. G., INTRODUCTION TO STRATIFICATIONAL LINGUISTICS, Harcourt Brace Jovanovich, Inc., New York, 1972.

24. Markowitz, H. M., Hausner, B., and Karr, H. W., SIMSCRIPT A SIMULATION PROGRAMMING LANGUAGE, Prentice-Hall, Englewood Cliffs, N. J., 1963.

25. McKeeman, W. M., Horning, J. J., Wortman, D. B., A COMPILER GENERATOR, Prentice-Hall, Englewood Cliffs, N.J., 1970.

26. Reich, P., "The relational network simulator," Linguistics Automation Project, Yale Univ., June 1968.

27. Rosen, S., PROGRAMMING SYSTEMS AND LANGUAGES, McGraw-Hill, New York, 1967.

28. Schank, R. C., "Conceptual dependency as a framework for linguistic analysis," LINGUISTICS 49, (June 1969), 28-50.

29. Schank, R. C., and Tesler, L. G., "A conceptual parser for natural language," In PROC. INT. JOINT CONF. ON ART. INTELL., (Walker and Norton, eds.), 1969, 569-578.

30. Schank, R. C., "Outline of a conceptual semantics for computer generation of coherent discourse," MATH BIOSCIENCES 5 (1969), 93-119.

31. Schank, R. C., Tesler, L. G., and Weber, S., "Spinoza II: Conceptual case-based natural language analysis," AI MEMO 109, Comp. Sci. Dept., Stanford Univ., Stanford, Calif., Jan. 1970.

32. Schank, R. C., Goldman, N., Rieger, C. J., and Riesbeck, C. K., "Primitive concepts underlying verbs of thought," AI MEMO 162, Comp. Sci. Dept., Stanford Univ., Stanford, Calif., Feb. 1972.

33. Schank, R. C., "Identification of conceptualizations underlying natural language," Comp. Sci. Dept., Stanford, Univ., Stanford, Calif., Feb. 1972.

34. Simmons, R. F.,"Answering English questions by computer: a survey," COMM. ACM 8, 1 (Jan. 1965), 53-70.

35. Simmons, R. F., Burger, J. F., and Schwarcz, R. M., "A computational model of verbal understanding," In PROC. AFIPS 1968 FJCC, Thompson Book Co., Washington D. C., 1968, 441-456.

36. Simmons, R. F., "Natural language question - answering systems: 1969," COMM. ACM 13, 1 (January 1970), 15-30.

37. Simmons, R. F., "Some semantic structures for representing English meanings, " Preprint, Dept. of Comp. Sci., Univ. of Texas, Austin, Texas, Nov. 1970.

38. Simmons, R. F., "Semantic networks: their computation and use for understanding English sentences," Technical Report NL-6, Dept. of Comp. Sci., Univ. of Texas, Austin, Texas, May 1972.

39. Simmons, R. F., and Slocum, J., "Generating English discourse from semantic networks," COMM. ACM 15, 10 (Oct. 1972), 891-905.

40. Thompson, F. B., Lockemann, P. C., Dostert, B., Deverill, R. S., "REL: a rapidly extensible language system," In PROC. 24th NAT'L CONF., ACM, N.Y., 1969, 399-417.

41. Weizenbaum, J., "Symmetric list processor, " COMM. ACM 6, 9 (Sept. 1963), 524-544.

42. Weizenbaum, J., "ELIZA - a computer program for the study of natural language communications between man and machine," COMM. ACM 9, 1 (Jan. 1966), 36-45.

43. Winograd, T., "Procedures as a representation for data in a computer program for understanding natural language," Technical Report AI TR-17, A. I. Lab., M. I. T., Cambridge, Mass., Feb. 1971.

44. Woods, W. A., "Transition network grammars for natural language analysis," COMM. ACM 13, 10 (Oct. 1970), 591-606.

45. Woods, W. A., Kaplan, R. M., Nash-Webber, B., "The lunar sciences natural language information system: final report," Technical Report No. 2378, Bolt Beranek and Newman, Inc., Cambridge, Mass., June 1972.

46. Yershov, A. P., "One view of man-machine interaction," J. ACM 12, 3 (July 1965), 315-325.

47. Yngve, V. H., "A model and an hypothesis for language structure," In PROC. AM. PHIL. SOC. 104, 5 (1960), 444-466.

48. Yngve, V. H., "Random generation of English sentences," PROC. 1961 CONF. MACH. TRANS., National Physical Laboratory Symposium No. 13, 1962, 66-80.

INITIAL DISTRIBUTION LIST

	No. Copies
Defense Documentation Center (DDC) Cameron Station Alexandria, Virginia 22314	20
Library Naval Postgraduate School Monterey, California 93940	2
Dean of Research Naval Postgraduate School Monterey, California 93940	2
Library Department of Operations Research and Administrative Sciences Naval Postgraduate School Monterey, California 93940	3
W. R. Church Computer Center Naval Postgraduate School Monterey, California 93940	2
Office of Naval Research Code 437 Information Systems Program Department of the Navy Arlington, Virginia 22217	2
Office of Naval Research 495 Summer Street Boston, Massachusetts 02210	1
Office of Naval Research 536 South Clark Street Chicago, Illinois 60605	1
Office of Naval Research 1030 East Green Street Pasadena, California 91101	1
Director, Naval Research Laboratory ATTN: Library, Code 2029 (ONRL) Washington, D. C. 20390	6
U. S. Naval Research Laboratory Code 2000 Technical Information Officer Washington, D. C. 20390	6

	No. Copies
Commandant of the Marine Corps (Code AX) Dr. A. L. Slafkosky Scientific Advisor Washington, D. C. 20380	1
Professor Alvin F. Andrus Department of Operations Research and Administrative Sciences Naval Postgraduate School Monterey, California 93940	1
LCDR Eldon S. Baker, USN Fleet Computer Programming Center Pacific San Diego, California 92147	1
Mr. A. M. Blum IBM Corporation 2651 Strang Boulevard Yorktown Heights, New York 10598	1
Mr. Daniel G. Bobrow Xerox Palo Alto Research Center 3180 Porter Drive Palo Alto, California 94304	1
Professor John Brown Department of Information and Computer Sciences University of California Irvine, California	1
LT Joseph L. Clapper, USNR Box 19A, R. D. #1 Wernersville, Pennsylvania 19565	1
Mr. Franklin S. Cooper Haskins Laboratories 270 Crown Street New Haven, Connecticut 06510	1
Professor Lewis G. Creary Department of Philosophy Case Western Reserve University Cleveland, Ohio 44106	1
Mr. Martin Dillon Department of Computer Science University of North Carolina Chapel Hill, North Carolina	1

	No. Copies
Mr. H. Robert Downs Systems Control, Inc. 260 Sheridan Avenue Palo Alto, California 94306	1
Mr. Horace Enea Artificial Intelligence Project Stanford University Stanford, California 94305	1
Mr. Martin Epstein Division of Computer Research and Technology National Institutes of Health Bethesda, Maryland 20014	1
Mr. Alfred M. Feiler Project TRANSIM Department of Engineering Engineering I, Room 3076 University of California Los Angeles, California 90024	1
Professor Robert B. Fetter Department of Administrative Sciences Yale University New Haven, Connecticut 06520	1
Professor Charles J. Fillmore Department of Linguistics University of California Berkeley, California	1
Professor George S. Fishman Department of Administrative Sciences Yale University New Haven, Connecticut 06520	1
Professor Joyce Friedman Department of Computer and Communication Sciences University of Michigan Ann Arbor, Michigan 48104	1
Professor Gregory D. Gibbons Mathematics Department Naval Postgraduate School Monterey, California 93940	1

	No. Copies
M. A. K. Halliday Center for Advanced Study in the Behavioral Sciences Stanford, California	1
LCDR Richard C. Hansen, USN 6265 Meadow Crest Drive La Mesa, California 92041	1
Professor David G. Hays Department of Linguistics State University of New York Buffalo, New York 14214	1
Charles L. Hedrick Carnegie-Mellon University Graduate School of Industrial Administration Schenley Park Pittsburgh, Pennsylvania 15213	1
Professor George E. Heidorn Department of Operations Research and Administrative Sciences Naval Postgraduate School Monterey, California 93940	8
MAJ Frederick H. Hemphill, Jr., USMC Computer Programming Branch SU1, MCTSSA MCAS(H) Santa Ana, California 92710	1
LT Bradley W. Hull, USN NAVSEC Center Bldg. Prince George's Center Hyattsville, Maryland 20782	1
Mr. Frederick Jelinek IBM Thomas J. Watson Research Center P. O. Box 218 Yorktown Heights, New York 10598	1
Professor Martin Kay Department of Information and Computer Sciences University of California Irvine, California	1

	No. Copies
Mr. Philip J. Kiviat Federal ADP Simulation Center Hoffman Bldg., Rm. 120 Alexandria, Virginia 22314	1
Professor Sheldon Klein Computer Sciences Department University of Wisconsin Madison, Wisconsin 53706	1
Professor Sydney M. Lamb Linguistics Department Yale University New Haven, Connecticut 06520	1
Bernard Lang Center for Research in Computing Technology Harvard University Cambridge, Massachusetts 02138	1
Professor David G. Lockwood Department of Linguistics Michigan State University Lansing, Michigan	1
LT Robert T. McGee, USN USS Harder (SS-568) F. P. O. San Francisco 96601	1
Professor Thomas O. Mitchell Graduate School Southern Illinois University Carbondale, Illinois 62901	1
CAPT Alfred H. Mossler, USMC MWCS -18, S-4 F. P. O. San Francisco 96602	1
Professor John Moyne Department of Computer Science Queens College Flushing, New York 11436	1
LT Kenneth S. Nelson, USN 1426 South Gourley Boise, Idaho 83705	1

	No. Copies
Mr. Jacques Noël Av. Observatoire 140 B-4000 LIEGE Belgium	1
Mr. Milos G. Pacak Division of Computer Research and Technology National Institutes of Health Bethesda, Maryland 20014	1
Mr. Arnold W. Pratt Division of Computer Research and Technology National Institutes of Health Bethesda, Maryland 20014	1
Professor Joseph Raben Computers and the Humanities Queens College of CUNY Flushing, New York 11367	1
Mr. Julian Reitman Preliminary Design Engineering Norden Division United Aircraft Corporation Norwalk, Connecticut 06856	1
LCDR John Rickelman, USN COMSUBLANT Staff Code N7 Norfolk, Virginia 23511	1
Mrs. Jane J. Robinson Mathematical Sciences Department IBM Thomas J. Watson Research Center P. O. Box 218 Yorktown Heights, New York 10598	1
Miss Jean E. Sammet Federal Systems Division IBM Corporation Cambridge, Massachusetts 02139	1
Professor Roger C. Schank Computer Science Department Stanford University Stanford, California 94305	1

	No. Copies
Professor Martin Shubik Department of Administrative Sciences Yale University New Haven, Connecticut 06520	1
Professor Robert F. Simmons Department of Computer Sciences University of Texas Austin, Texas 78712	1
Mrs. Dana Small NELC San Diego, California 92152	1
Professor Stanley Y. W. Su Center for Informatics Research University of Florida Gainesville, Florida 32601	1
Mrs. Carol Thompson IBM Thomas J. Watson Research Center P. O. Box 218 Yorktown Heights, New York 10598	1
Professor Frederick Thompson California Institute of Technology Pasadena, California	1
Mr. Heath Tuttle Route 6, Box 141 Chapel Hill, North Carolina 27514	1
Mr. Donald Walker Artificial Intelligence Center Stanford Research Institute Menlo Park, California 94024	1
Mr. Yorick Wilks Artificial Intelligence Project Stanford University Stanford, California 94305	1
LCDR Robert J. Williams, USN Navy Finance Center New Federal Office Building Cleveland, Ohio 44199	1
Professor Terry Winograd Artificial Intelligence Laboratory Massachusetts Institute of Technology Cambridge, Massachusetts 02139	1

	No. Copies
Mr. William A. Woods Bolt, Beranek and Newman, Inc. 50 Moulton Street Cambridge, Massachusetts 02138	1
Professor David Jefferson Code 53Df Naval Postgraduate School Monterey, California 93940	1

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Naval Postgraduate School Monterey, California 93940	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP
3. REPORT TITLE Natural Language Inputs to a Simulation Programming System	
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) Technical Report, 1972	
5. AUTHOR(S) (First name, middle initial, last name) George E. Heidorn	

6. REPORT DATE October 1972	7a. TOTAL NO. OF PAGES 388	7b. NO. OF REFS 48
8a. CONTRACT OR GRANT NO. b. PROJECT NO. PO 1-0177 c. Identifying Number NR 049-314 d.	9a. ORIGINATOR'S REPORT NUMBER(S) NPS-55HD72101A 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
10. DISTRIBUTION STATEMENT Approved for public release; distribution unlimited.		
11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Information Systems Program, Office of Naval Research	

13. ABSTRACT

This report describes research which has been done toward developing a system for performing simulation analyses through natural language interaction with a computer. A general system for natural language processing, consisting of an IBM 360 FORTRAN program and a "rule language", has been developed. The user of this system must write sets of "decoding" and "encoding" rules, along with some declarations, to specify how processing is to be done for his application. Decoding rules specify how text is to be processed to produce an entity-attribute-value data structure, and encoding rules specify how text is to be produced from such a data structure. The FORTRAN program does the processing according to the sets of rules it is given. Several sets of decoding and encoding rules have been written to implement a specific system which is capable of carrying on a dialogue in English about a simple queuing problem and then producing a program in the GPSS language to do the simulation.

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Digital Computer Simulation						
Simulation Programming						
GPSS						
Queuing Problems						
Computational Linguistics						
Natural Language						
Stratificational Grammar						
Grammar Rule Language						
Mechanical Translation						
Semantics						
Artificial Intelligence						